Building bioinformatic pipelines

6/20/2019
What is a pipeline?

A *pipeline* or *workflow* refers to a series of processing steps such that output of each process is the input of the next, typically done to transform raw data into something more interpretable.
Why bother building pipelines?

1. Reproducibility
2. Data provenance
3. Automation
4. Transparency
Pipelines aid in reproducibility

Reproducibility = obtaining the same result* using the same code and data

*within reason (e.g., some aligners assign multi-mapping reads to a random location)
Data provenance contextualizes results

Provenance refers to the description of the origin of a piece of data

• The steps taken to arrive at a piece of data
• The software used
• The version of the software used
• The arguments supplied to the software used
Automation: the amount of data keeps increasing
Automation: some pipelines complex
Simple alignment pipeline with bowtie2

# align reads with bowtie2
bowtie2 -x ref.fa –U short_read.fq > aln-se.sam

# convert from sam to bam
samtools view -bS aln-se.sam > aln-se.bam

# sort bam file
samtools sort aln-se.bam > aln-se.sorted.bam
Simple sample script

#!/usr/bin/env bash

## tools
BOWTIE=/usr/local/bin/bowtie2 #v2.3.5.1
SAMTOOLS=/usr/local/bin/samtools #v1.9

## reference genome
REFERENCE=/usr/local/ref/e_coli.fa

$BOWTIE -x $REFERENCE -U A.fastq.gz > A.sam
$SAMTOOLS view -bS A.sam > A.bam
$SAMTOOLS sort A.bam > A.sorted.bam
For loops

#!/usr/bin/env bash

BOWTIE=/usr/local/bin/bowtie2
SAMTOOLS=/usr/local/bin/samtools
REFERENCE=/usr/local/ref/e_coli.fa

for read in $(ls *fastq.gz) ; do
    $BOWTIE -x $REFERENCE -U $read > ${read/.fastq.gz/.sam}
    $SAMTOOLS view -bS ${read/.fastq.gz/.sam} > ${read/.fastq.gz/.bam}
    $SAMTOOLS sort ${read/.fastq.gz/.bam} > ${read/.fastq.gz/.sorted.bam}
done
GNU parallel

tool for processing repetitive commands

parallel [options] [command [arguments]] ::: <files>

• ::: <files> or find <files> |

• The file name: {}

• The file name with the extension removed: {.}
  e.g. test.fa would become test

• --jobs, -j n
GNU parallel pipeline

THREADS=2

parallel --jobs $THREADS gunzip {} ::: *fastq.gz

parallel --jobs $THREADS $BOWTIE -x $REFERENCE -U {} "">" {}.sam ::: *fastq

parallel --jobs $THREADS $SAMTOOLS view -bS {}.sam "">" {}.bam ::: *sam

parallel --jobs $THREADS $SAMTOOLS sort {}.sam "">" {}.sorted.bam ::: *bam
A brief history of make

• first introduced by Stuart Feldman in 1977 at Bell Labs
• build automation tool
  • used to build executable programs and libraries from source code
  • however, make is not limited to building binaries and libraries
Key features of make

• Dependency analysis
• Re-entrancy
• Parallelization
• Pattern rules / abstraction
• Audit trail
what is make?

*make* is a program that reads a makefile and that builds one or more files from zero or more other files that they depend on.
how does make do what it does?

make parses the makefile, builds a dependency tree (by determining the relationships between the inputs and outputs), and then traverses each branch of the tree, executing commands along the way.
what is a makefile?

a *makefile* is a text file which contains *rules* for how to create a set of target files.
what is a rule?

a rule tells *make* which series of commands to execute and what files must exist beforehand in order to create a set of targets from some input.
the general form of a rule is:

target ... : dependency ... 
   command

...
a practical example: alignment

BOWTIE=/usr/local/bin/bowtie2 #v2.3.5.1
SAMTOOLS=/usr/local/bin/samtools #v1.9
REFERENCE=/usr/local/ref/e_coli.fa

all: A.sam

A.sam: A.fastq.gz
  $(BOWTIE) -x $(REFERENCE) -U A.fastq.gz > A.sam
adding another step:

BOWTIE=/usr/local/bin/bowtie2 #v2.3.5.1
SAMTOOLS=/usr/local/bin/samtools #v1.9
REFERENCE=/usr/local/ref/e_coli.fa

all: A.bam

A.sam: A.fastq.gz
   $(BOWTIE) -x $(REFERENCE) -U A.fastq.gz > A.sam

A.bam: A.sam
   $(SAMTOOLS) view -bS A.sam > A.bam
automatic variables

BOWTIE=/usr/local/bin/bowtie2 #v2.3.5.1
SAMTOOLS=/usr/local/bin/samtools #v1.9
REFERENCE=/usr/local/ref/e_coli.fa

clean: clean.A

A.bam: A.sam
  $(SAMTOOLS) view -bS $< > $@

A.sam: A.fastq.gz
  $(BOWTIE) -x $(REFERENCE) -U $< > $@
using pattern rules: the percent sign

% : roughly equivalent to * in a Unix shell
- represents any number of any characters
- can be placed anywhere within pattern
- can only occur once

some valid uses:
  %.v
  s%.o
  wrapper_%
- characters other than % match literally within a filename
revisiting alignment...

FASTQFILES := $(wildcard *.fastq.gz)

all: $(FASTQFILES:.fastq.gz=.sorted.bam)

%.sam: %.fastq.gz
    $(BOWTIE) -x $(REFERENCE) -U A.fastq.gz > A.sam

%.bam: %.sam
    $(SAMTOOLS) view -bS $< > $@

%.sorted.bam: %. bam
    $(SAMTOOLS) sort $< > $@
visualizing the dependency tree
the -j switch

-\texttt{j [jobs]}, \texttt{--jobs[=jobs]}
  specifies the number of jobs (commands) to run simultaneously.
why make? the limits of a script:

1. linear execution
   • make -j

2. truncated files
   • .DELETE_ON_ERROR:

3. unable to resume
   • make

4. poor audit trail
   • make -nB > make.log
Limitations of make

• Wasn’t designed for bioinformatic analyses
• Syntax requires understanding rule structure
• Lacks support for multiple outputs from single command
• No support for multiple wildcards per name
• No built-in support for distributed computing
Ways to parallelize

Single computer, single core

Single computer, multiple cores

Multiple computers, multiple cores

Image From: http://cloudcomputingnet.com/category/cloudcomputing/grid-computing/
Future trends

COMMON WORKFLOW LANGUAGE

Singularity containers

Image From: https://www.hpcwire.com/2017/05/04/singularity-hpc-container-technology-moves-lab/#foo/3/0/Singularity-architecture_G-Kurtzer-e1477021972985.jpg
Many contemporary alternatives to make

https://github.com/pditommaso/awesome-pipeline
#!/usr/bin/env cwl-runner

cwlVersion: v1.0
class: CommandLineTool
baseCommand: echo
inputs:
  message:
    type: string
inputBinding:
  position: 1
outputs: []

message: Hello world!

$ cwl-runner 1st-tool.cwl echo-job.yml
[job 1st-tool.cwl] /tmp/tmpmM5S_1$ echo \n  'Hello world!'
Hello world!
[job 1st-tool.cwl] completed success
{}
Final process status is success
Pipelines tip of iceberg concerning reproducibility

From: Experimenting with reproducibility in bioinformatics
Yang-Min Kim, Jean-Baptiste Poline, Guillaume Dumas
bioRxiv 143503; doi: https://doi.org/10.1101/143503