Unlocking the Power of Genomic Analysis in Julia

Edmund Miller

July 26th, 2023

Overview

About Me

- Phd Candidate @ University of Texas at Dallas
- nf-core maintainer

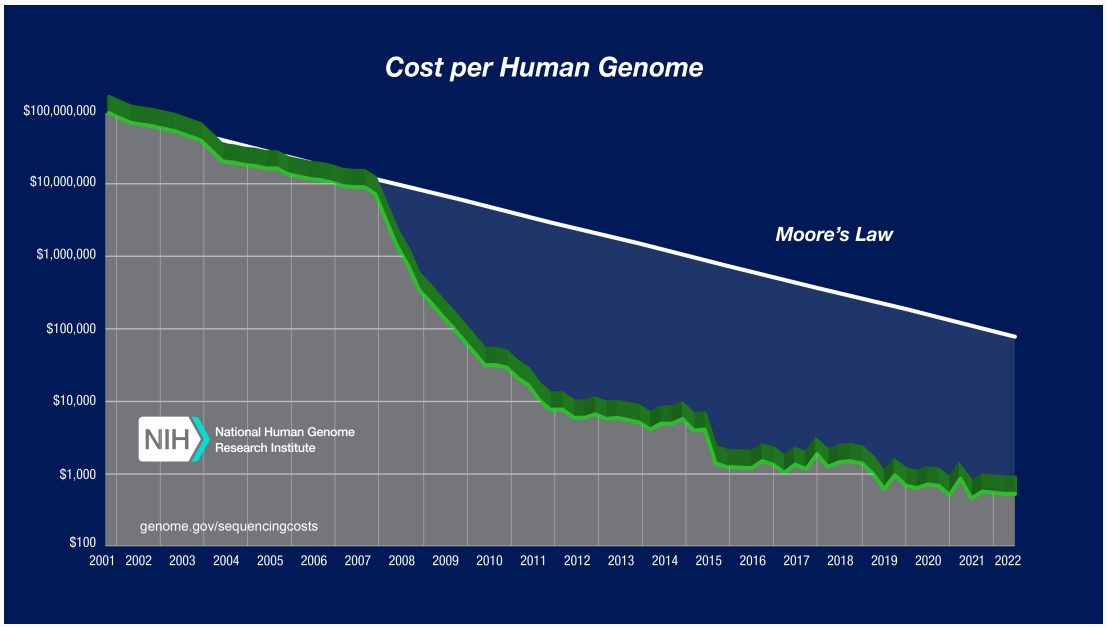
Why should biologists working in Genomics be interested in Julia?

- "Why We Created Julia"[1]
  - As easy for statistics as R
  - With mathematical notation like Matlab
  - As usable for general programming as Python
  - As natural for string processing as Perl
  - As good at gluing programs together as the shell
  - As fast as C
- Reproducibility

[1] `https://julialang.org/blog/2012/02/why-we-created-julia`

Why should Julia enthusiasts interested in Genomics?

- Interest in keeping costs low through more efficient computation
  - Cost of Whole Genome Sequencing: ~$100-1000
  - Cost of computational Analysis: ~$25



Overview

- Julia Features for Analysis
- Ecosystem
- Julia in Workflows

Julia Features for Analysis

Juliaup

```
curl -fsSL https://install.julialang.org | sh
```

- Cross-platform installer
- Install specific Julia versions
- Stay up to date with release channels (Stable, LTS)

Package management - Pkg.jl

```
# ]
(@v1.8) pkg>
(@v1.8) pkg> add Example
Resolving package versions...
Installed Example — v0.5.3
    Updating
     ↪  `~/.julia/environments/v1.8/Project.toml`
[7876af07] + Example v0.5.3
    Updating
     ↪  `~/.julia/environments/v1.8/Manifest.toml`
[7876af07] + Example v0.5.3
julia> import Example

julia> Example.hello("friend")
"Hello, friend"
```

Pkg.jl Environments

```
(@v1.8) pkg> activate rnaseq-analysis
[ Info: activating new environment at
 ↪  `~/rnseq-analysis/Project.toml`.
(rnaseq-analysis) pkg>
(rnaseq-analysis) pkg> status
    Status `~/rnseq-analysis/Project.toml`
(empty environment)
(rnaseq-analysis) pkg> add GFF3 GenomicFeatures
...
(rnaseq-analysis) pkg> status
  Status `~/rnaseq-analysis/Project.toml`
  [af1dc308] GFF3 v0.2.3
^ [899a7d2d] GenomicFeatures v2.1.0
```

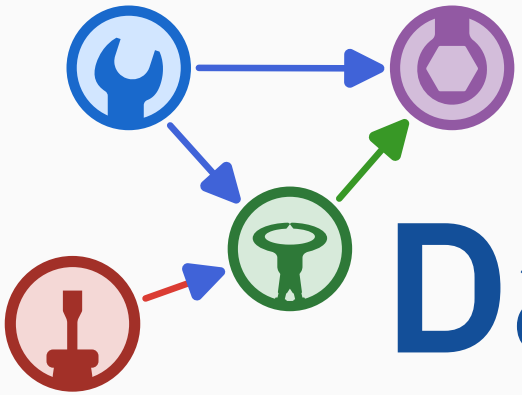
Want to hack on a project?

```
pkg> develop --local Bed
```

There's a full clone at `dev/Bed`!

Other niceties

- PkgTemplates.jl - Easy Package Creation
- Julia REPL Mastery Workshop
- VS Code - Batteries included environment



DataToolkit.jl

DataToolkit - Example declarative data set

```toml
[[HNSC_Phenotypes]]
uuid = "c2f8275e-f5b7-46f5-a95c-af3835573258"
description = "TCGA Head and Neck Cancer (HNSC)
 ↪  RNA-seq data"

    [[HNSC_Phenotypes.storage]]
    driver = "web"
    checksum = "crc32c:d5c06b86"
    url =
     ↪  "https://tcga-xena-hub.s3.us-east-1.amazonaws.c
```

DataToolkit - Example declarative data set

```
[[HNSC_Phenotypes.loader]]
driver = "csv"
type = "DataFrame"

    [HNSC_Phenotypes.loader.args]
    delim = "\t"
    header = 1
    select = ["sampleID", "sample_type"]
    types = "String"
```

DataToolkit - Features in this example

- A named dataset `[[iris]]`
- Which can be uniquely identified `uuid = "..."`
- With metadata `description = "..."`
- Named storage/loader backends `driver = "web"`
- Content verification `checksum = "crc32c:d5c06b86"`
- Storage/loader arguments `url = "...", args.header = 1`

DataToolkit - Using a dataset in computation

DataToolkit - Using a dataset in computation

```
julia> using DataToolkit, DataFrames

julia> mean(d"HNSC_Phenotypes::Matrix", dims=1)
1×5 Matrix{Float64}:
 5.84333  3.05733  3.758  1.19933  1.0
```

DataToolkit - Using a dataset in computation

```
julia> using DataToolkit, DataFrames

julia> mean(d"HNSC_Phenotypes::Matrix", dims=1)
1×5 Matrix{Float64}:
 5.84333  3.05733  3.758  1.19933  1.0
```

More than just string matching for types:

```
julia> mean(d"HNSC_Phenotypes::Array{T<:Any, 2}",
 ↪  dims=1)
1×5 Matrix{Float64}:
 5.84333  3.05733  3.758  1.19933  1.0
```

DataToolkit - Want to learn more?

Robust data management made simple: Introducing DataToolkit

Timothy Chapman

Friday, 07-28, 16:00–16:30 (US/Eastern), 32-123

Ecosystem

Package comparisons - General Utilities

Purpose	Python	R	Julia
Plotting	Matplotlib	ggplot2	PyPlot.jl / Makie.jl / Gadfly.jl
Dataframes	Pandas/Polars	tibble	DataFrames.jl

Package comparisons - Biological File Formats

Purpose	Python	R	Julia
Sam/Bam files	Bio.SeqIO	Rsamtools	XAM.jl
Fastq files	Bio.SeqIO	ShortRead	FASTX.jl
Variants/vcf	PyVCF	vcfR	GeneticVariation.jl / VariantCallFormat.jl
	Bio.Phylo		Phylogenies.jl
	Bio.PDB		BioStructures.jl
	BCBio.GFF		GFF3.jl

Package comparisons - Genomic Analysis

Purpose	Python	R	Julia
Genomic Ranges	pyranges / pybedtools	rtracklayer	GenomicFeatures.jl
Genomic Ranges		GenomicsRanges	Intervals
Blast	Bio.Blast	metablastr	BioTools.jl
DNA/RNA/AA	Bio.SeqIO		BioSequences.jl
Data Retrieval	pyranges / biomart (api)	biomartr	BioServices.jl / BioFetch.jl
Genomic Annotations	pyranges / pybedtools	GenomicsRanges	GenomicAnnotations.jl
Population Genetics	Bio.PopGen		PopGen.jl

What about when you can’t replace popular packages?

What about when you can't replace popular packages?

- Working in Python & R is like buying a house in DFW

What about when you can't replace popular packages?

- Working in Python & R is like buying a house in DFW
- Downtown, constant re-development(pip, poetry, hatch, piptools, conda)

What about when you can't replace popular packages?

- Working in Python & R is like buying a house in DFW
- Downtown, constant re-development(pip, poetry, hatch, piptools, conda)
- Compared to the suburbs where you need a car

What about when you can't replace popular packages?

- Working in Python & R is like buying a house in DFW
- Downtown, constant re-development(pip, poetry, hatch, piptools, conda)
- Compared to the suburbs where you need a car
- Urban Sprawl of Python & R packages
  - DESeq2/edgeR/seurat/scanpy
  - ggplot2

What about when you can't replace popular packages?

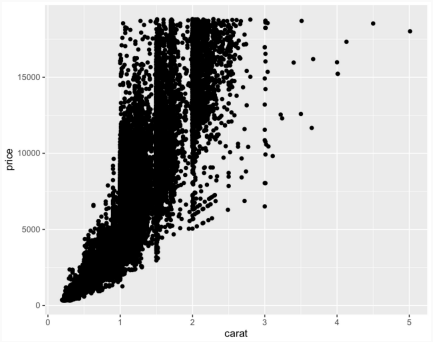
Options:

- JuliaInterop · GitHub
- RCall.jl

- PythonCall.jl
- Calling commandline tools from Julia

JuliaInterop - RCall

```
julia> using RCall
# type `$`
R> install.packages("ggplot2")
R> library(ggplot2)
R> data(diamonds)
R> ggplot(diamonds, aes(x=carat, y=price)) \
    + geom_point()
```



PythonCall - Python

```python
import scanpy as sc


def preprocessing(adata):
    # Perform preprocessing of a anndata object
    sc.pp.filter_cells(adata, min_genes=200)
    sc.pp.filter_genes(adata, min_cells=3)

    # Normalization and scaling:
    sc.pp.normalize_total(adata, target_sum=1e4)
    sc.pp.log1p(adata)
    x = adata.X
    data = tf.data.Dataset.from_tensor_slices((x,
    ↪  x))
    return data, x
```

PythonCall - Julia

```
using PythonCall

sc = pyimport("scanpy")

function preprocessing(adata)
    sc.pp.filter_cells(adata, min_genes=200)
    sc.pp.filter_genes(adata, min_cells=3)

    # Normalization and scaling:
    sc.pp.normalize_total(adata, target_sum=1e4)
    sc.pp.log1p(adata)

    x = pyconvert(Array{Float32}, adata.X)
    return [x, x], x
end
```

PythonCall and Pycall are different

- Doesn't have to support as much legacy
  - PythonCall supports Julia 1.6.1+ and Python 3.7+
  - PyCall supports Julia 0.7+ and Python 2.7+.
- Uses CondaPkg by default
- You can use them both at the same time if you needed to for some reason

Managing conda envs in Julia

```
using Conda, RCall

Conda.add("bioconductor-deseq2",
 ↪  channel="bioconda", :rnaseq)
```

CondaPkg.jl

```
julia> using CondaPkg
pkg> conda add_channel bioconda
pkg> conda add bioconductor-deseq2
```

CondaPkg.toml

```
channels = ["conda-forge"]

[deps]
bioconductor-deseq2 = ""
```

Calling commandline tools from Julia

```
julia> mycommand = `echo hello`
`echo hello`

julia> typeof(mycommand)
Cmd

julia> run(mycommand);
hello
```

Docs on Running External Programs

Calling commandline tools from Julia

```
julia> files = ["read_1.fastq", "read_2.fastq"]
2-element Vector{String}:
"read_1.fastq"
"read_2.fastq"

julia> `bwa mem $files`
`bwa mem 'read_1.fastq read_2.fastq'`
```

Packages plugging directly in Common packages

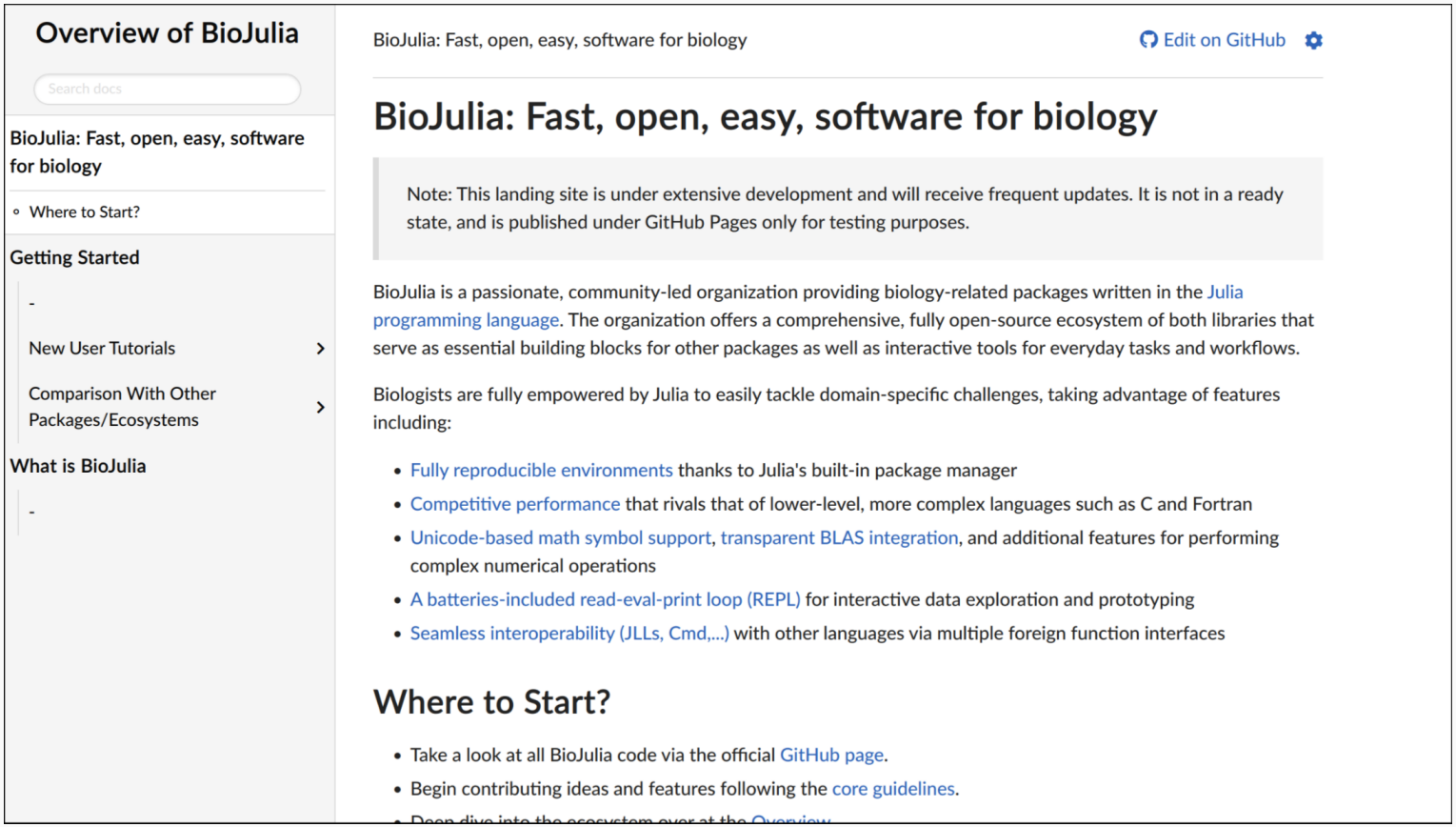
```
using FileIO, BedgraphFiles, DataTables,
 ↪  IndexedTables, Gadfly

# Load into a DataTable
dt = DataTable(load("data.bedgraph"))

# Load into an IndexedTable
it = IndexedTable(load("data.bedgraph"))
# Plot directly with Gadfly
plot(load("data.bedgraph"), xmin=:leftposition,
 ↪  xmax=:rightposition, y=:value, Geom.bar)

load("data.bedgraph") |> @filter(_.chrom ==
 ↪  "chr19") |> save("data-chr19.bedgraph")
```

BioJuliaDocs



BioJuliaDocs

Julia in Workflows

Running Julia in Snakemake

```
from snakemake.remote import AUTO
iris =
↪  "https://raw.githubusercontent.com/scikit-learn/scik
rule calling_script:
    input:
        AUTO.remote(iris)
    output:
        "results/out.csv",
    container: "docker://julia"
    script:
        "bin/smk_script.jl"
```

In the Julia script, a snakemake object is available, which can be accessed similar to the Python case, with the only difference that you have to index from 1 instead of 0.

Running Julia in Snakemake - Inside the Julia script

```
import Pkg; Pkg.add(["CSV", "DataFrames"])

using CSV, DataFrames

df = DataFrame(CSV.File(snakemake.input[1],
 ↪  footerskip=50))
names(df)
CSV.write(snakemake.output[1], df)

do_something(snakemake.input[1],
 ↪  snakemake.output[2], snakemake.threads,
 ↪  snakemake.config["myparam"])
```

Running Julia in Nextflow - Installing Packages to Julia Depot

Julia Lang, Docker & Nextflow | Personal Homepage of Alex Peltzer

```
// nextflow.config
env {
    JULIA_DEPOT_PATH = "/usr/local/share/julia"
}
```

Running Julia in Nextflow - The Nextflow script

```
process shebang {
    container 'julia'

    input:
    path csv_file

    output:
    path "out.csv"

    """
    #!/usr/bin/env -S julia --startup-file=no

    using CSV, DataFrames
    df = DataFrame(CSV.File($csv_file,
     ↪  footerskip=50))
    CSV.write("out.csv", df)
    """
```

Running Julia in Nextflow - The Nextflow script

```
process cli {
    container 'julia'

    input:
    path csv_file

    output:
    stdout

    """
    julia hello.jl $csv_file
    """
}
```

Running Julia in Nextflow - The Julia script

```
#!/usr/bin/env -S julia --color=yes
 ↪  --startup-file=no

println(PROGRAM_FILE);
abspath(PROGRAM_FILE) == @__FILE__

@show ARGS

for x in ARGS
    println(x)
end
```

- Move it to the `bin/` folder of the pipeline, and make it executable (`chmod +x bin\*.jl`)

Analysis Example

Overlapping BED files

```
# overlap of H3K27ac and P63 peaks identifies
 ↪  enhancer regions where p63 binds
using Downloads

if !isfile(raw"H3K27ac.consensus_peaks.bed")

    ↪  Downloads.download("https://utdallas.box.com/sh
    ↪  "H3K27ac.consensus_peaks.bed")

    ↪  Downloads.download("https://utdallas.box.com/sh
    ↪  "p63_4A4.consensus_peaks.bed")
end

using GenomicFeatures
using BED
```

Overlapping BED files

```
# Create an interval collection in memory.
h3k27ac_icol = open(BED.Reader,
 ↪  "H3K27ac.consensus_peaks.bed") do reader
    IntervalCollection(reader)
end

p63_icol = open(BED.Reader,
 ↪  "p63_4A4.consensus_peaks.bed") do reader
    IntervalCollection(reader)
end
```

Overlapping BED files

```
overlap_icol = eachoverlap(h3k27ac_icol,
↪  p63_icol)
first(overlap_icol)
overlaps = collect(overlap_icol)
overlaps
BED.Record(p63_icol)
```

Overlapping BED files

```
writer = BED.Writer(output)
expected_entries = BED.Record[]
for interval in open(BED.Reader, filename)
    write(writer, interval)
    push!(expected_entries, interval)
end
```

Conclusion

Where is Julia lacking?

- Creating binaries/CLIs
- But what about Rust?
  - Rust for tools
  - Julia for analysis

Resources

- GitHub - BioJulia/BioTutorials: Tutorial Notebooks of BioJulia
- New Documenter.jl Docs!

Slides

link.edmundmiller.dev