title	author	date
Automating Data-analysis Pipelines	Shaun Jackman	2014-11-03

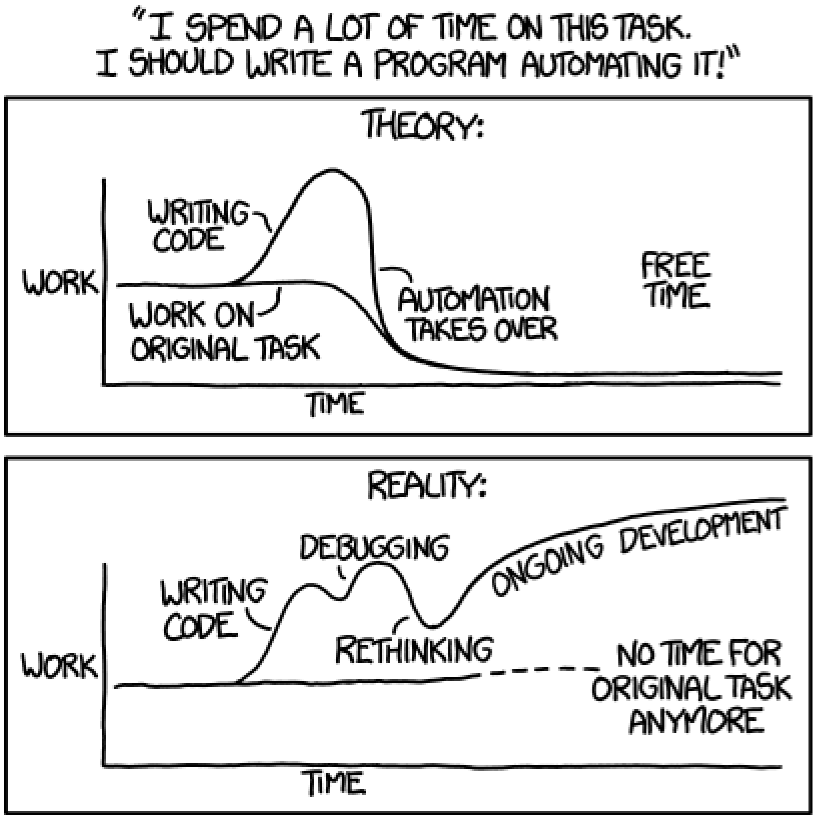
Automating Data-analysis Pipelines

| UBC STAT 545A/STAT 547M | 2014-11-03 | Shaun Jackman @sjackman | Jenny Bryan @JennyBryan |

Pipelines Automation Dependencies

Automation

- using R
- using the shell and Rscript
- using make



'Automating' comes from the roots 'auto-' meaning 'self-', and 'mating', meaning 'screwing'.

Pipelines

A pipeline

breaks up a monolithic make-all-the-things script into discrete, manageable chunks.

Each stage of the pipeline

| … defines its input and its outputs. | … does *not* modify its inputs, so it is *idempotent*.

| Rerunning a stage of the pipeline | produces the same results as the previous run.

Advantage #1

| When you modify one stage of the pipeline, | you don't have to rerun the entire pipeline.

You only rerun the downstream, dependent stages.

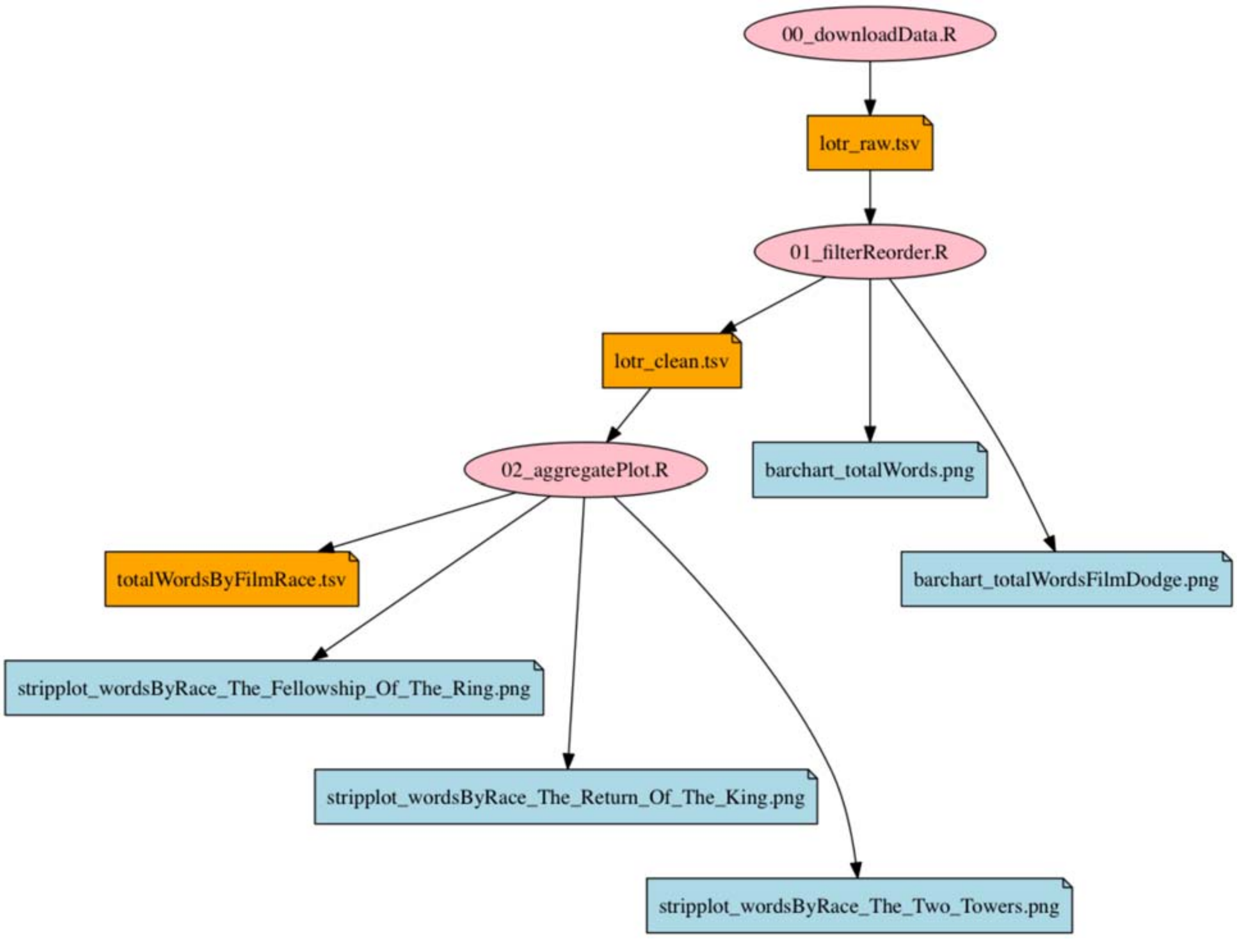
Advantage #2

Divide up work amongst a group by assigning to each person stages of the pipeline design.

Advantage #3

| You can draw pretty pictures of your pipeline, | because a pipeline is a graph.

|


00_downloadData.R
lotr_raw.tsv
01_filterReorder.R
lotr_clean.tsv
02_aggregatePlot.R
barchart_totalWords.png
totalWordsByFilmRace.tsv
barchart_totalWordsFilmDodge.png
stripplot_wordsByRace_The_Fellowship_Of_The_Ring.png
stripplot_wordsByRace_The_Return_Of_The_King.png
stripplot_wordsByRace_The_Two_Towers.png

| *01_justR*

Automation

Automate a pipeline

| ... to reproduce previous results. | ... to recreate results deleted by fat fingers. | ... to rerun the pipeline with updated software. | ... to run the same pipeline on a new data set.

Tools

- R
- the shell and Rscript
- make

R

```
#!/usr/bin/env Rscript
source("00_downloadData.R")
source("01_filterReorder.R")
source("02_aggregatePlot.R")
```

. . .

- Shows in what order to run the scripts.
- You can resume the pipeline from the middle.

Shell and Rscript

```
#!/bin/sh
set -eux
Rscript 00_downloadData.R
Rscript 01_filterReorder.R
Rscript 02_aggregatePlot.R
```

. . .

Allows you to easily run your pipeline from the shell.

. . .

Option	Effect
`set -e`	Stop at the first error
`set -u`	Undefined variables are an error
`set -x`	Print each command as it is run

Mix R scripts with other tools

```
#!/bin/sh
set -eux
curl -L http://bit.ly/lotr_raw-tsv >lotr_raw.tsv
```

```
Rscript 01_filterReorder.R
Rscript 02_aggregatePlot.R
```

. . .

R is a good tool, but not always the best tool for the job.

Not sacrilege, but the principal tenet of a polyglot.

Makefile

```
#!/usr/bin/make -f

lotr_raw.tsv:
	curl -L http://bit.ly/lotr_raw-tsv >lotr_raw.tsv

lotr_clean.tsv: 01_filterReorder.R lotr_raw.tsv
	Rscript 01_filterReorder.R

totalWordsByFilmRace.tsv: 02_aggregatePlot.R lotr_clean.tsv
	Rscript 02_aggregatePlot.R
```

. . .

| A Makefile gives both the commands | *and their dependencies*.

Make is beautiful

| Tell Make how to create one type of file from another | and which files you want to create.

. . .

| Make looks at which files you have | and figures out how to create the files that you want.

Dependency graph

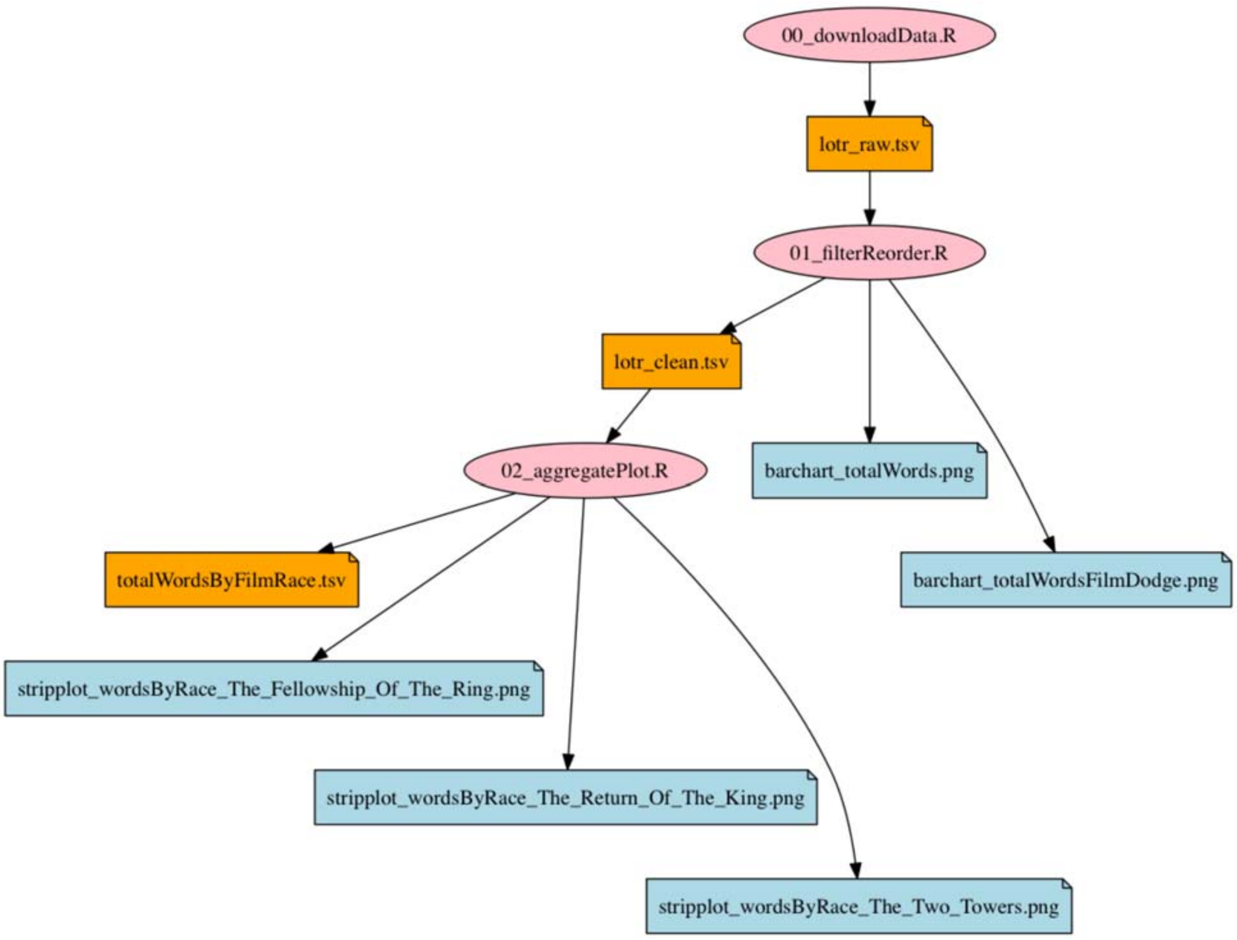
A pipeline is a graph

Scripts and data files are vertices of the graph.

Dependencies between stages are edges of the graph.

Both scripts and data files are shown.

|

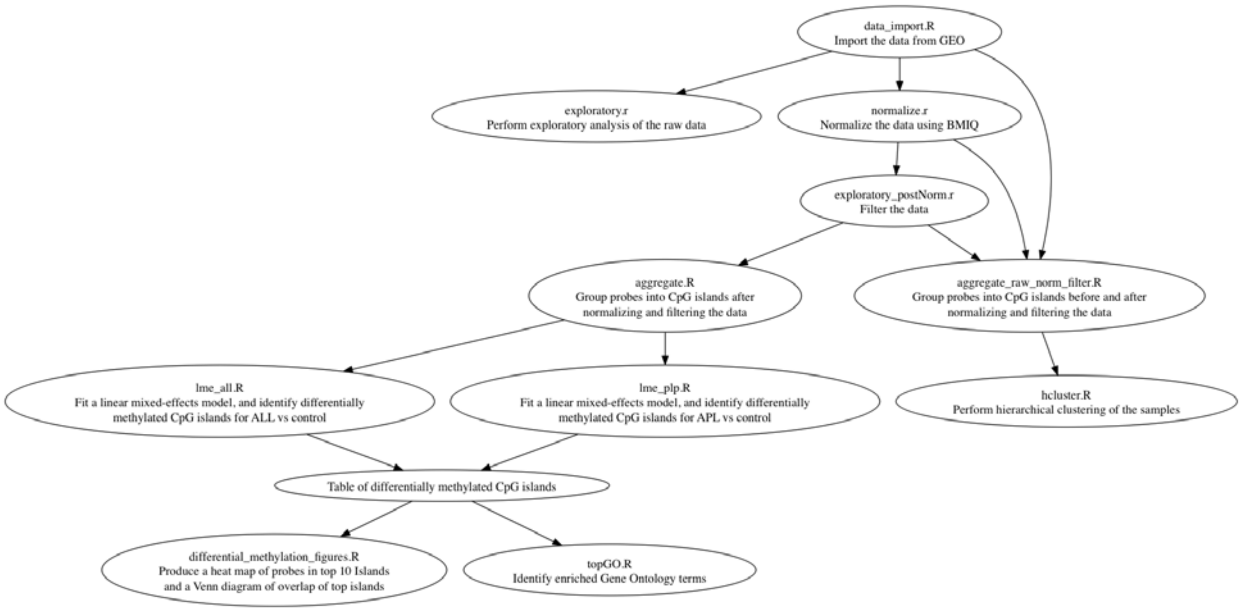


00_downloadData.R
lotr_raw.tsv
01_filterReorder.R
lotr_clean.tsv
02_aggregatePlot.R
barchart_totalWords.png
totalWordsByFilmRace.tsv
barchart_totalWordsFilmDodge.png
stripplot_wordsByRace_The_Fellowship_Of_The_Ring.png
stripplot_wordsByRace_The_Return_Of_The_King.png
stripplot_wordsByRace_The_Two_Towers.png

| *01_justR*

- Only dependencies between scripts are shown.
- Data files are not shown.
- Run the scripts in *topographical* order.

|


data_import.R
Import the data from GEO
exploratory.r
Perform exploratory analysis of the raw data
normalize.r
Normalize the data using BMIQ
exploratory_postNorm.r
Filter the data
aggregate.R
Group probes into CpG islands after normalizing and filtering the data
aggregate_raw_norm_filter.R
Group probes into CpG islands before and after normalizing and filtering the data
lme_all.R
Fit a linear mixed-effects model, and identify differentially methylated CpG islands for ALL vs control
lme_plp.R
Fit a linear mixed-effects model, and identify differentially methylated CpG islands for APL vs control
hcluster.R
Perform hierarchical clustering of the samples
Table of differentially methylated CpG islands
differential_methylation_figures.R
Produce a heat map of probes in top 10 Islands and a Venn diagram of overlap of top islands
topGO.R
Identify enriched Gene Ontology terms

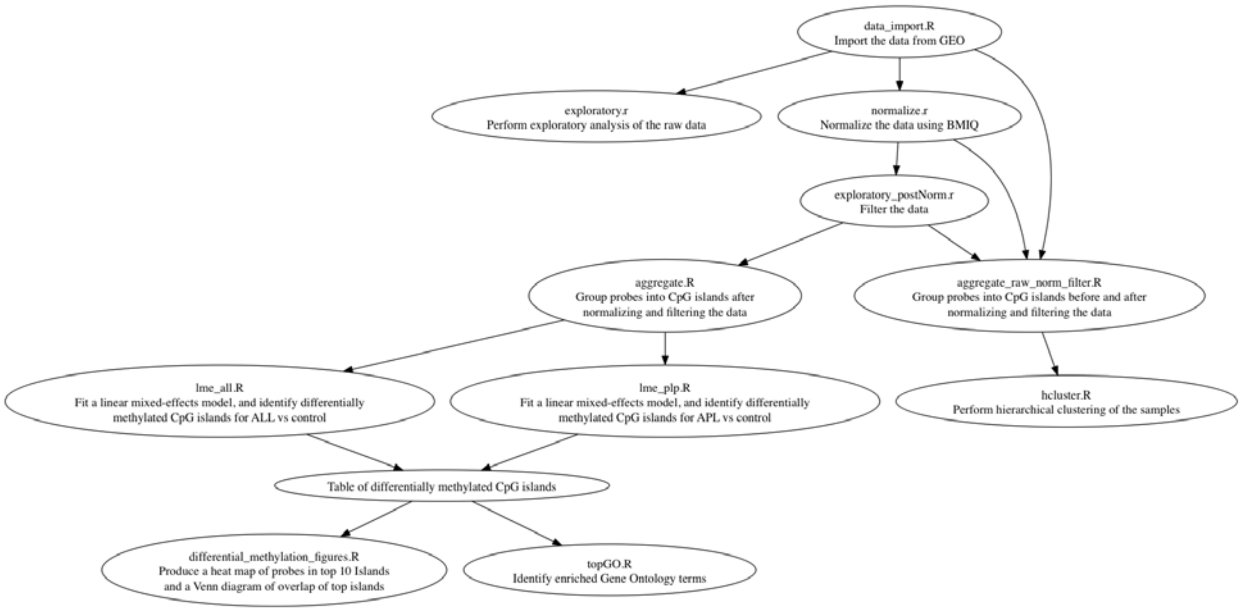
| *STAT 540 Differential Methylation in Leukemia*

Order of dependencies

A shell script gives *one* order in which you can successfully run the pipeline.

. . .

Unless the pipeline is completely linear, there are likely other such orders.


data_import.R
Import the data from GEO
exploratory.r
Perform exploratory analysis of the raw data
normalize.r
Normalize the data using BMIQ
exploratory_postNorm.r
Filter the data
aggregate.R
Group probes into CpG islands after normalizing and filtering the data
aggregate_raw_norm_filter.R
Group probes into CpG islands before and after normalizing and filtering the data
lme_all.R
Fit a linear mixed-effects model, and identify differentially methylated CpG islands for ALL vs control
lme_plp.R
Fit a linear mixed-effects model, and identify differentially methylated CpG islands for APL vs control
hcluster.R
Perform hierarchical clustering of the samples
Table of differentially methylated CpG islands
differential_methylation_figures.R
Produce a heat map of probes in top 10 Islands and a Venn diagram of overlap of top islands
topGO.R
Identify enriched Gene Ontology terms

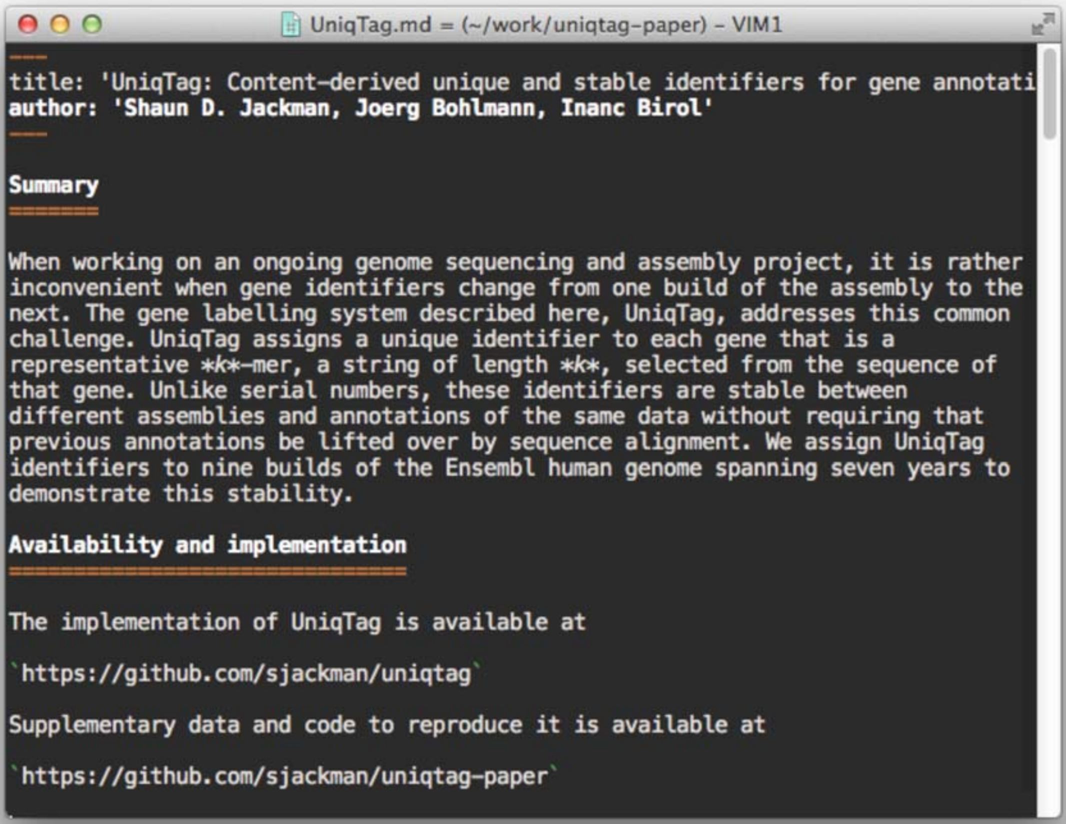
A different order of commands may be more convenient, but without information of the dependencies, you're stuck with the given order.

A reproducible manuscript

One Makefile

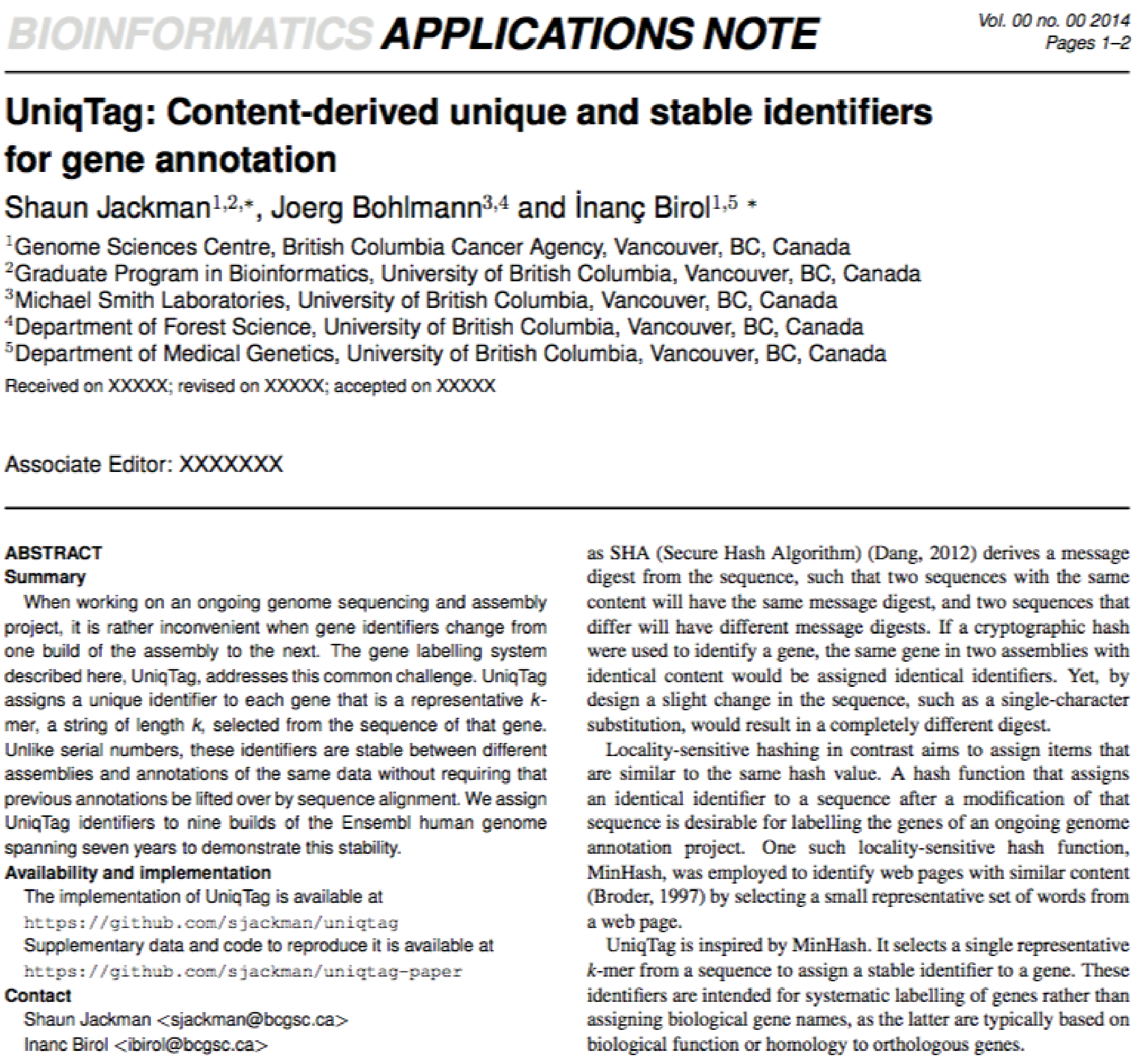
- Downloads the data
- Runs the command-line programs
- Performs the statistical analyses using R
- and Generates the TSV tables
- Renders the figures using ggplot2
- Renders the supplementary material using RMarkdown
- Renders the manuscript using Pandoc

Turns this



```
---
title: 'UniqTag: Content-derived unique and stable identifiers for gene annotati
author: 'Shaun D. Jackman, Joerg Bohlmann, Inanc Birol'
---

Summary
=======

When working on an ongoing genome sequencing and assembly project, it is rather
inconvenient when gene identifiers change from one build of the assembly to the
next. The gene labelling system described here, UniqTag, addresses this common
challenge. UniqTag assigns a unique identifier to each gene that is a
representative *k*-mer, a string of length *k*, selected from the sequence of
that gene. Unlike serial numbers, these identifiers are stable between
different assemblies and annotations of the same data without requiring that
previous annotations be lifted over by sequence alignment. We assign UniqTag
identifiers to nine builds of the Ensembl human genome spanning seven years to
demonstrate this stability.

Availability and implementation
===============================

The implementation of UniqTag is available at

`https://github.com/sjackman/uniqtag`

Supplementary data and code to reproduce it is available at

`https://github.com/sjackman/uniqtag-paper`
```

Into this


BIOINFORMATICS APPLICATIONS NOTE Vol. 00 no. 00 2014 Pages 1–2

UniqTag: Content-derived unique and stable identifiers for gene annotation

Shaun Jackman[1,2,*], Joerg Bohlmann[3,4] and İnanç Birol[1,5 *]
[1]Genome Sciences Centre, British Columbia Cancer Agency, Vancouver, BC, Canada
[2]Graduate Program in Bioinformatics, University of British Columbia, Vancouver, BC, Canada
[3]Michael Smith Laboratories, University of British Columbia, Vancouver, BC, Canada
[4]Department of Forest Science, University of British Columbia, Vancouver, BC, Canada
[5]Department of Medical Genetics, University of British Columbia, Vancouver, BC, Canada

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT
Summary
When working on an ongoing genome sequencing and assembly project, it is rather inconvenient when gene identifiers change from one build of the assembly to the next. The gene labelling system described here, UniqTag, addresses this common challenge. UniqTag assigns a unique identifier to each gene that is a representative *k*-mer, a string of length *k*, selected from the sequence of that gene. Unlike serial numbers, these identifiers are stable between different assemblies and annotations of the same data without requiring that previous annotations be lifted over by sequence alignment. We assign UniqTag identifiers to nine builds of the Ensembl human genome spanning seven years to demonstrate this stability.
Availability and implementation
The implementation of UniqTag is available at
`https://github.com/sjackman/uniqtag`
Supplementary data and code to reproduce it is available at
`https://github.com/sjackman/uniqtag-paper`
Contact
Shaun Jackman <sjackman@bcgsc.ca>
Inanc Birol <ibirol@bcgsc.ca>

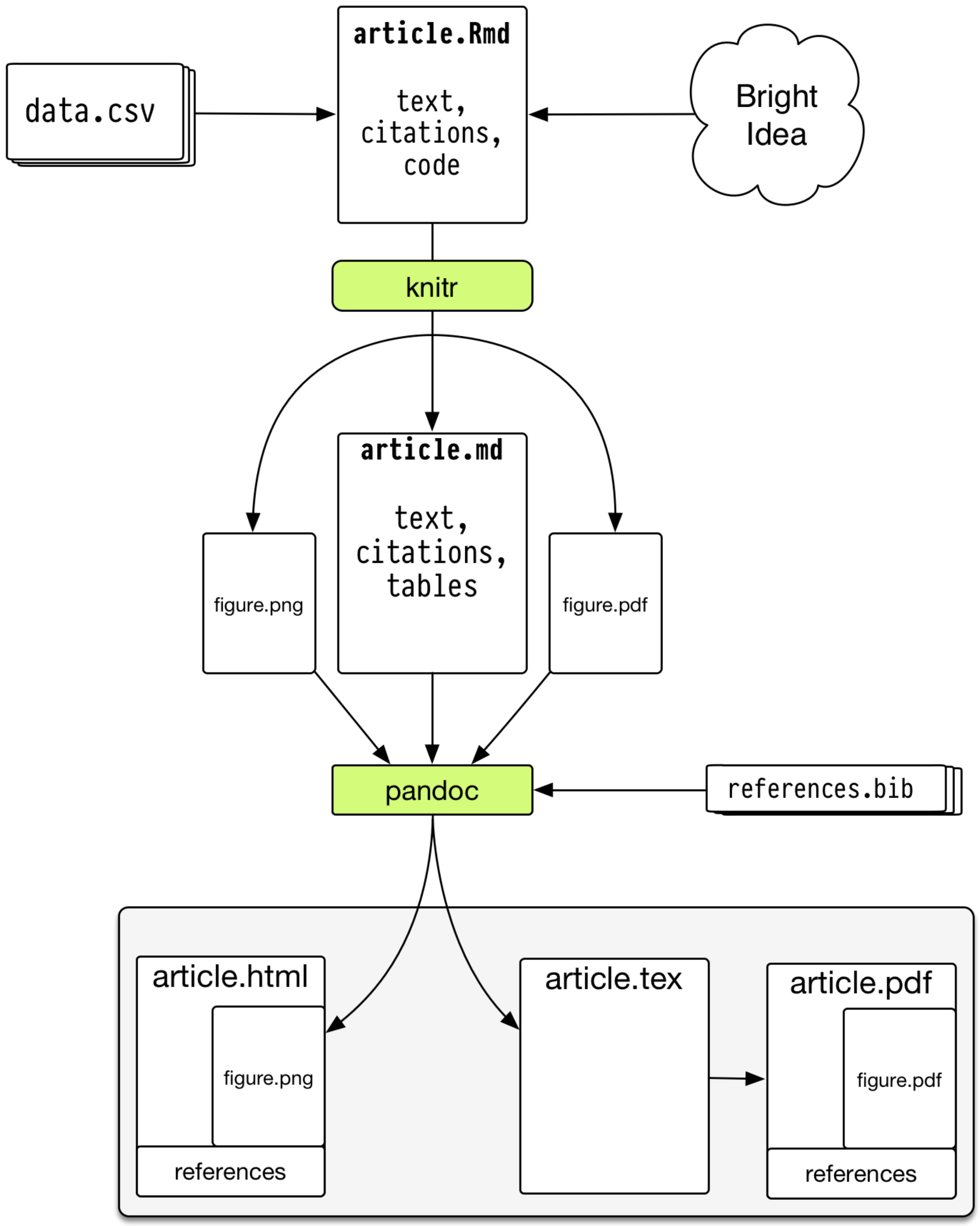
as SHA (Secure Hash Algorithm) (Dang, 2012) derives a message digest from the sequence, such that two sequences with the same content will have the same message digest, and two sequences that differ will have different message digests. If a cryptographic hash were used to identify a gene, the same gene in two assemblies with identical content would be assigned identical identifiers. Yet, by design a slight change in the sequence, such as a single-character substitution, would result in a completely different digest.

Locality-sensitive hashing in contrast aims to assign items that are similar to the same hash value. A hash function that assigns an identical identifier to a sequence after a modification of that sequence is desirable for labelling the genes of an ongoing genome annotation project. One such locality-sensitive hash function, MinHash, was employed to identify web pages with similar content (Broder, 1997) by selecting a small representative set of words from a web page.

UniqTag is inspired by MinHash. It selects a single representative *k*-mer from a sequence to assign a stable identifier to a gene. These identifiers are intended for systematic labelling of genes rather than assigning biological gene names, as the latter are typically based on biological function or homology to orthologous genes.

Workflow

Plain Text, Papers, Pandoc by Kieran Healy



Markdown for the manuscript

Markdown is a plain-text typesetting language

```
A header
========

A list:
```

```
+ This text is *italic*
+ This text is **bold**
```

A header

A list:

- This text is *italic*
- This text is **bold**

RMarkdown

- RMarkdown interleaves prose with R code
  - to aggregate and summarize the data
  - to generate tables
  - to render figures using ggplot2
- RMarkdown is ideal for supplementary material

RMarkdown example

~~~
The Sum of 1 + 1
================

The sum of 1 + 1 is calculated as follows.

```{r}
1 + 1
```

![*Fig. 1*: A graphical view of 1 + 1](figure.png)
~~~

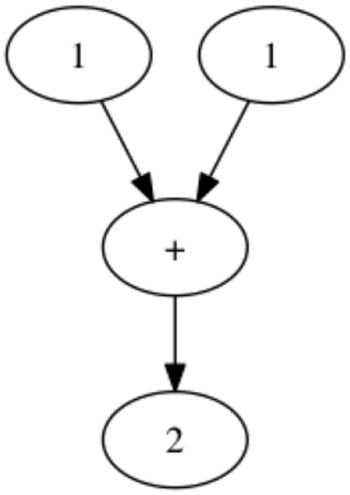
article.Rmd

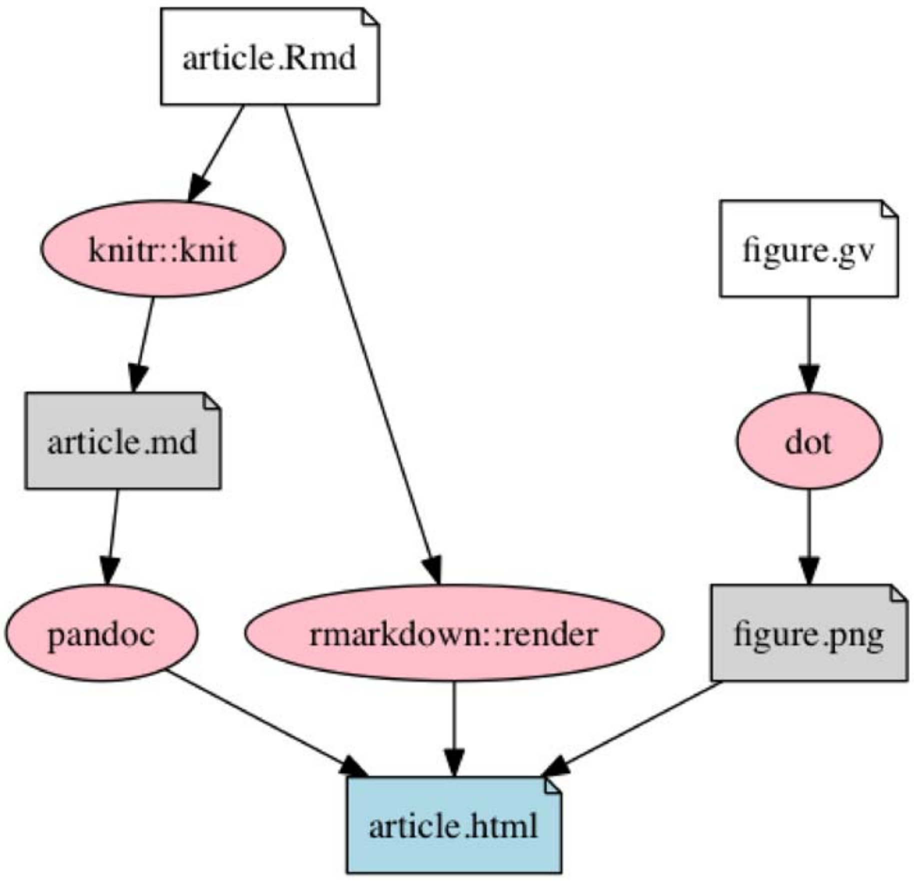
The Sum of 1 + 1

The sum of 1 + 1 is calculated as follows.

```
1 + 1
```

```
## [1] 2
```





Dependencies of *article/Makefile*

Render HTML

```
%.md: %.Rmd
        Rscript -e 'knitr::knit("$<", "$@")'

%.html: %.md
        pandoc -s -o $@ $<

%.html: %.Rmd
        Rscript -e 'rmarkdown::render("$<")'
```

```
article.html: figure.png

%.png: %.gv
	dot -Tpng $< >$@
```

```
make article.html
```

```
dot -Tpng figure.gv >figure.png
Rscript -e 'rmarkdown::render("article.Rmd")'
```

Knit Markdown

```
%.md: %.Rmd
	Rscript -e 'knitr::knit("$<", "$@")'

%.html: %.md
	pandoc -s -o $@ $<

%.html: %.Rmd
	Rscript -e 'rmarkdown::render("$<")'

article.html: figure.png

%.png: %.gv
	dot -Tpng $< >$@
```

```
make article.md article.html
```

```
Rscript -e 'knitr::knit("article.Rmd", "article.md")'
dot -Tpng figure.gv >figure.png
pandoc -s -o article.html article.md
```

Pandoc

| Pandoc renders attractive documents and slides | from plain-text typesetting formats

It converts between every format known (just about)

- Markdown
- HTML
- LaTeX

- PDF
- ODT and docx (yes, really)

Evolving a Makefile

```
#!/bin/sh
set -eux
dot -Tpng -o figure.png figure.gv
Rscript -e 'knitr::knit("article.Rmd")'
pandoc -s -o article.html article.md
```

Shell script

```
all:
	dot -Tpng -o figure.png figure.gv
	Rscript -e 'knitr::knit("article.Rmd")'
	pandoc -s -o article.html article.md
```

First Makefile

```
all: article.html

article.html:
	dot -Tpng -o figure.png figure.gv
	Rscript -e 'knitr::knit("article.Rmd")'
	pandoc -s -o article.html article.md
```

Add a rule to build `article.html`

```
all: article.html

article.html: article.Rmd
	dot -Tpng -o figure.png figure.gv
	Rscript -e 'knitr::knit("article.Rmd")'
	pandoc -s -o article.html article.md
```

`article.html` depends on `article.Rmd`

```
all: article.html

figure.png: figure.gv
	dot -Tpng -o figure.png figure.gv

article.md: article.Rmd
	Rscript -e 'knitr::knit("article.Rmd")'

article.html: article.md figure.png
	pandoc -s -o article.html article.md
```

Split one rule into three

```
all: article.html

figure.png: figure.gv
	dot -Tpng -o $@ $<

article.md: article.Rmd
	Rscript -e 'knitr::knit("$<", "$@")'

article.html: article.md figure.png
	pandoc -s -o $@ $<
```

Use the variables `$<` and `$@` for the input and output file

```
all: article.html

%.png: %.gv
	dot -Tpng -o $@ $<

%.md: %.Rmd
	Rscript -e 'knitr::knit("$<", "$@")'

article.html: article.md figure.png
	pandoc -s -o $@ $<
```

Use pattern rules. The `%` matches any string

```
all: article.html

%.png: %.gv
```

```
        dot -Tpng -o $@ $<

%.md: %.Rmd
        Rscript -e 'knitr::knit("$<", "$@")'

%.html: %.md
        pandoc -s -o $@ $<

article.html: figure.png
```

`article.html` also depends on `figure.png`

```
all: article.html

clean:
        rm -f article.md article.html figure.png

%.png: %.gv
        dot -Tpng -o $@ $<

%.md: %.Rmd
        Rscript -e 'knitr::knit("$<", "$@")'

%.html: %.md
        pandoc -s -o $@ $<

article.html: figure.png
```

Add the target named `clean`

```
all: article.html

clean:
        rm -f article.md article.html figure.png

.PHONY: all clean
.DELETE_ON_ERROR:
.SECONDARY:

%.png: %.gv
        dot -Tpng -o $@ $<

%.md: %.Rmd
        Rscript -e 'knitr::knit("$<", "$@")'

%.html: %.md
        pandoc -s -o $@ $<
```

```
article.html: figure.png
```

Add `.PHONY`, `.DELETE_ON_ERROR` and `.SECONDARY`

```
all: article.html

clean:
	rm -f article.md article.html figure.png

.PHONY: all clean
.DELETE_ON_ERROR:
.SECONDARY:

# Render a GraphViz file
%.png: %.gv
	dot -Tpng -o $@ $<

# Knit a RMarkdown document
%.md: %.Rmd
	Rscript -e 'knitr::knit("$<", "$@")'

# Render a Markdown document to HTML
%.html: %.md
	pandoc -s -o $@ $<

# Dependencies on figures
article.html: figure.png
```

fin

Links

| STAT 545A | xkcd automation | R | Rscript | shell | make | Markdown | RMarkdown | Pandoc | ggplot2 | Plain Text, Papers, Pandoc | STAT 540 Differential Methylation in Leukemia

@sjackman | github.com/sjackman | sjackman.ca

Shaun Jackman

| Genome Sciences Centre, BC Cancer Agency | Vancouver, Canada | @sjackman | github.com/sjackman | sjackman.ca