A bioinformatic pipeline for GWAS

Christian Werner

(Quantitative geneticist and biostatistician) **EiB, CIMMYT**, Texcoco (Mexico)

Filippo Biscarini — HerrFalloppio

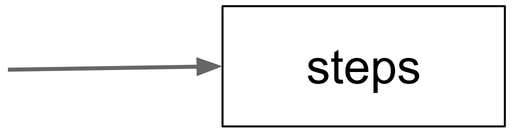
(Biostatistician, bioinformatician and quantitative geneticist) **CNR-IBBA**, Milan (Italy)

Oscar González-Recio — OscarGenomics

(Computational biologist and quantitative geneticist) **INIA-UPM**, Madrid (Spain)

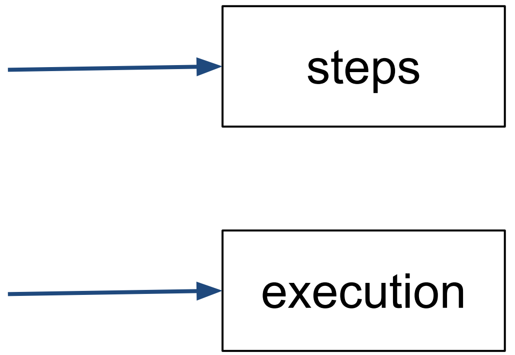
bioinformatic analysis

- typical bioinformatic analyses involve a **series of data transformations / operations** (e.g. joining paired-end reads, assembling reads into longer sequences, aligning sequences to a reference genome, calling variants etc.)



bioinformatic analysis

- typical bioinformatic analyses involve a **series of data transformations / operations** (e.g. joining paired-end reads, assembling reads into longer sequences, aligning sequences to a reference genome, calling variants etc.)
- to be run **sequentially** or **in parallel** on one or **multiple samples** / input files



bioinformatic analysis

- typical bioinformatic analyses involve a **series of data transformations / operations** (e.g. joining paired-end reads, assembling reads into longer sequences, aligning sequences to a reference genome, calling variants etc.) $\rightarrow$ $\boxed{\text{steps}}$
- to be run **sequentially** or **in parallel** on one or **multiple samples** / input files $\rightarrow$ $\boxed{\text{execution}}$

this calls for a **structured** / organized **pipeline** / workflow of analysis

bioinformatic analysis

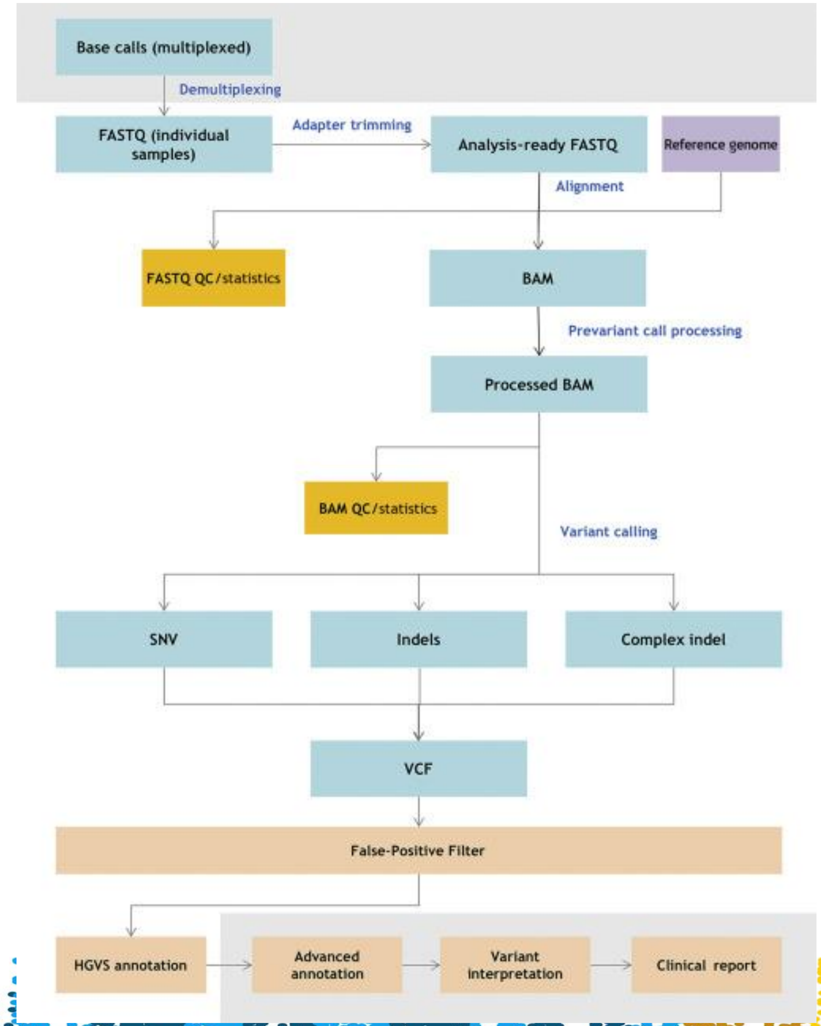
- large datasets
- embarrassing parallelization (hundreds of jobs)
- many different tools, scripts, languages (dependencies) → difficult to maintain and to update, almost impossible to make it portable

this calls for a **structured** / organized **pipeline** / workflow of analysis

bioinformatic pipelines

- **chaining** together multiple consecutive **steps** (processes) of a data analysis problem/project

bioinformatic pipelines



(Roy et al, 2018)

bioinformatic pipelines

- **chaining** together multiple consecutive **steps** of a data analysis problem/project
- advantages**:**
    - **reproducibility**
    - **modularity**
    - **parallelization**
    - **scalability**
    - **portability**
    - **usability**
    - **automation**

bioinformatic pipelines - reproducibility

Reproducibility: quantum mechanics (probabilistic world) → average (statistical) reproducibility (besides, different environments / different labs → different results are expected); this is true also in bioinformatics (same pipeline, same data, different machines → different results are possible: underlying variance in the libraries, floats, machines -e,g, Mac cs Linux- and so on …)

- https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0080278;
- https://www.nature.com/articles/nbt.3820

bioinformatic pipelines - predecessors

- stand-alone **scripts**: e.g. Unix shell, Python etc.
    - features: variables, conditional logic
    - limitations: no support for dependencies and reentrancy (e.g. concatenated steps, adding input samples, updating parameters/resources, recovering from local failures, run/start intermediate steps etc.)

bioinformatic pipelines - predecessors

- stand-alone **scripts**: e.g. Unix shell, Python etc.
    - features: variables, conditional logic
    - limitations: no support for dependencies and reentrancy (e.g. concatenated steps, adding input samples, updating parameters/resources, recovering from local failures, run/start intermediate steps etc.)
- **Make**: compilation / build automation tool [Stallman and McGrath, 2020]
    - use an instruction file (Makefile) to generate a target (e.g. compiled software)
    - features: wildcards, dependency tree

bioinformatic pipelines - predecessors

- stand-alone **scripts**: e.g. Unix shell, Python etc.
  - features: variables, conditional logic
  - limitations: no support for dependencies and reentrancy (e.g. concatenated steps, adding input samples, updating parameters/resources, recovering from local failures, run/start intermediate steps etc.)
- **Make**: compilation / build automation tool
  - use an instruction file (Makefile) to generate a target (e.g. compiled software)
  - wildcards = “card” (string) used to represent any other “card” (string),

    e.g. `ls -alt *.map` (lists all map files in folder)

    * stands for any prefix before .map (it is a wildcard)

bioinformatic pipelines - predecessors

- stand-alone **scripts**: e.g. Unix shell, Python etc.
  - features: variables, conditional logic
  - limitations: no support for dependencies and reentrancy (e.g. concatenated steps, adding input samples, updating parameters/resources, recovering from local failures, run/start intermediate steps etc.)
- **Make**: compilation / build automation tool
  - use an instruction file (Makefile) to generate a target (e.g. compiled software)
  - dependency tree: file modification datetimes are used to determine which steps are required to generate the target

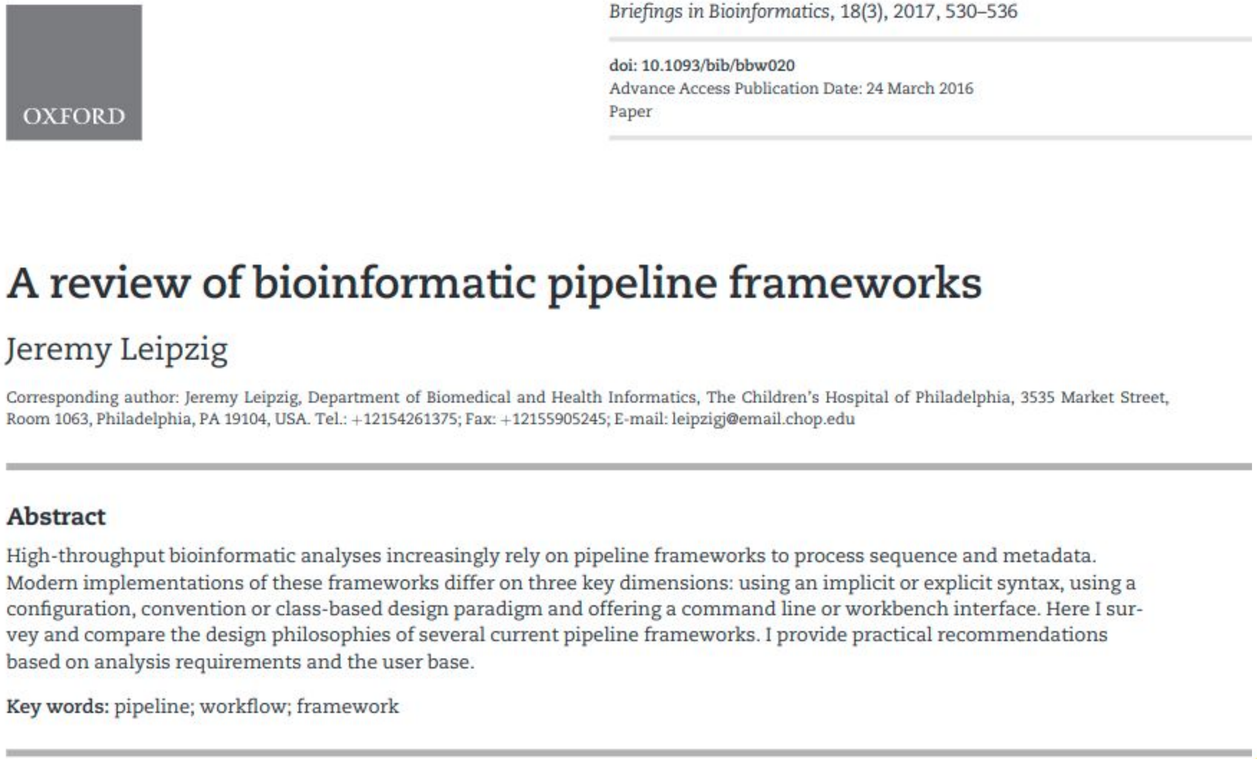
bioinformatic pipelines - predecessors

- stand-alone **scripts**: e.g. Unix shell, Python etc.
  - features: variables, conditional logic
  - limitations: no support for dependencies and reentrancy (e.g. concatenated steps, adding input samples, updating parameters/resources, recovering from local failures, run/start intermediate steps etc.)
- **Make**: compilation / build automation tool
  - use an instruction file (Makefile) to generate a target (e.g. compiled software)
  - features: wildcards, dependency tree
  - limitations: not designed for scientific pipelines, no direct support for distributed/parallel computing, insufficient flexibility for complex parameters, input data, execution logic etc.

bioinformatic pipelines - available frameworks

- pipeline execution engines / workflow management systems
    - **Snakemake:** https://snakemake.readthedocs.io/en/stable/
    - **Nextflow:** https://www.nextflow.io/
    - **BigDataScript:** https://pcingola.github.io/BigDataScript/
    - **Pipengine:** https://github.com/fstrozzi/bioruby-pipengine

bioinformatic pipelines - reading a bit more



OXFORD

Briefings in Bioinformatics, 18(3), 2017, 530–536

doi: 10.1093/bib/bbw020
Advance Access Publication Date: 24 March 2016
Paper

A review of bioinformatic pipeline frameworks

Jeremy Leipzig

Corresponding author: Jeremy Leipzig, Department of Biomedical and Health Informatics, The Children's Hospital of Philadelphia, 3535 Market Street, Room 1063, Philadelphia, PA 19104, USA. Tel.: +12154261375; Fax: +12155905245; E-mail: leipzigj@email.chop.edu

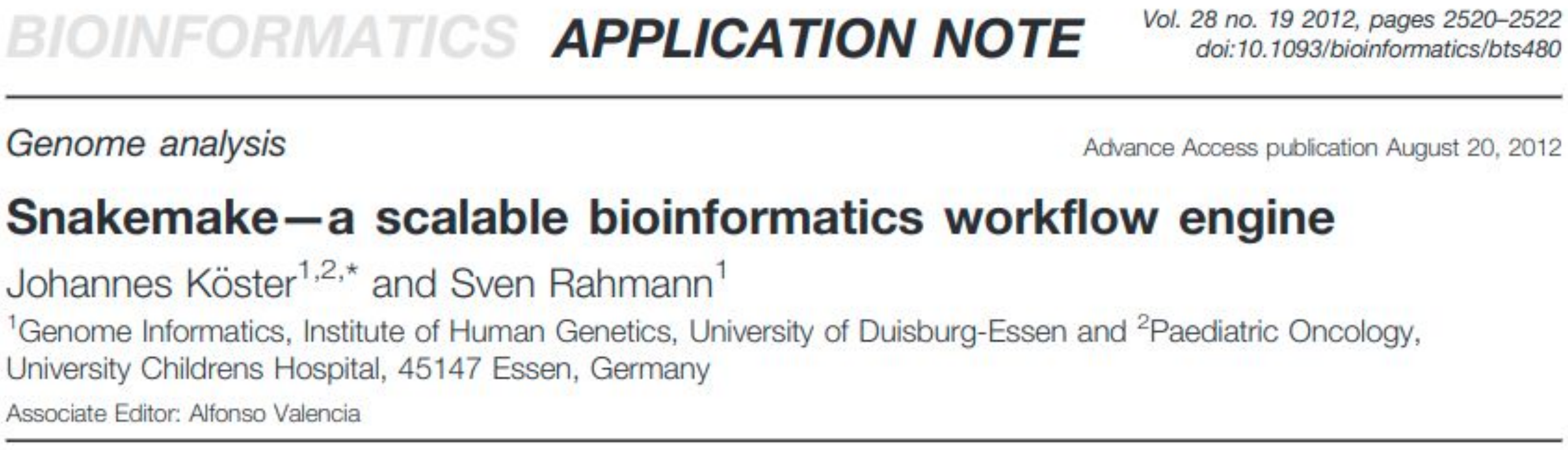
Abstract

High-throughput bioinformatic analyses increasingly rely on pipeline frameworks to process sequence and metadata. Modern implementations of these frameworks differ on three key dimensions: using an implicit or explicit syntax, using a configuration, convention or class-based design paradigm and offering a command line or workbench interface. Here I survey and compare the design philosophies of several current pipeline frameworks. I provide practical recommendations based on analysis requirements and the user base.

Key words: pipeline; workflow; framework

Snakemake

bioinformatic pipelines - Snakemake


BIOINFORMATICS APPLICATION NOTE

Vol. 28 no. 19 2012, pages 2520–2522
doi:10.1093/bioinformatics/bts480

Genome analysis

Advance Access publication August 20, 2012

Snakemake—a scalable bioinformatics workflow engine

Johannes Köster[1,2,*] and Sven Rahmann[1]

[1]Genome Informatics, Institute of Human Genetics, University of Duisburg-Essen and [2]Paediatric Oncology, University Childrens Hospital, 45147 Essen, Germany

Associate Editor: Alfonso Valencia

- Pythonic variant of GNU Make
- Makefile → **Snakefile**

bioinformatic pipelines - Snakemake

- pipelines are defined in terms of **rules** that define how to create output files from input files:

  input → [rule] → output

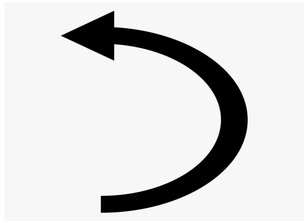
- **dependencies** between the rules are determined automatically, creating a DAG (directed acyclic graph) of jobs that can be automatically parallelized

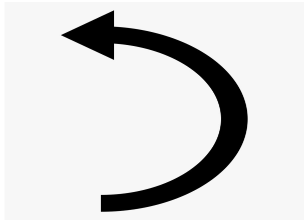
Snakemake - the snakefile

```
## target rule: files that we want to generate
## as final output of the pipeline
rule targets:
    input:
        "snp_sorted.map.gz"

## step 2
## rule to compress the sorted map file
rule gzip:
    input:
        "snp_sorted.map"
    output:
        "snp_sorted.map.gz"
    shell:
        "gzip {input}"

## step 1
## rule to sort the map file
rule sort:
    input:
        "snp.map"
    output:
        "snp_sorted.map"
    shell:
        "sort -n -k1 {input} > {output}"
```

Snakemake - the snakefile

```
## target rule: files that we want to generate
## as final output of the pipeline
rule targets:
    input:
        "snp_sorted.map.gz"

## step 2
## rule to compress the sorted map file
rule gzip:
    input:
        "snp_sorted.map"
    output:
        "snp_sorted.map.gz"
    shell:
        "gzip {input}"

## step 1
## rule to sort the map file
rule sort:
    input:
        "snp.map"
    output:
        "snp_sorted.map"
    shell:
        "sort -n -k1 {input} > {output}"
```





[if exists(snp_sorted.map.gz)] else:

[if snp_sorted.map.gz > snp_sorted.map] else:

[if snp_sorted.map > snp.map] else:

Snakemake - the snakefile

```
## target rule: files that we want to generate
## as final output of the pipeline
rule targets:
    input:
        "snp_sorted.map.gz"

## step 2
## rule to compress the sorted map file
rule gzip:
    input:
        "snp_sorted.map"
    output:
        "snp_sorted.map.gz"
    shell:
        "gzip {input}"

## step 1
## rule to sort the map file
rule sort:
    input:
        "snp.map"
    output:
        "snp_sorted.map"
    shell:
        "sort -n -k1 {input} > {output}"
```

keywords: rule, input, output, shell
wildcards: {}, {input}, {output}

Snakemake - execution

```
snakemake --help

snakemake --dryrun --snakefile example_snakefile
```

Snakemake - execution

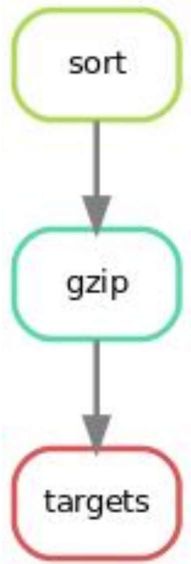
```
snakemake --help

snakemake --dryrun --snakefile example_snakefile

snakemake -n -s example_snakefile
```

[have a look at the log from the dry run]

short option names

Snakemake - the DAG

```
snakemake --dag --dryrun --snakefile
example_snakefile | dot -Tsvg > dag_example.svg
```



Snakemake - execution

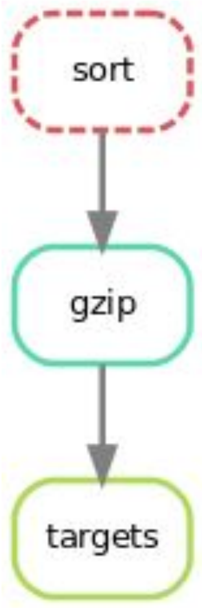
```
snakemake --snakefile example_snakefile --cores 1
```

- true run
- check output

Snakemake - execution

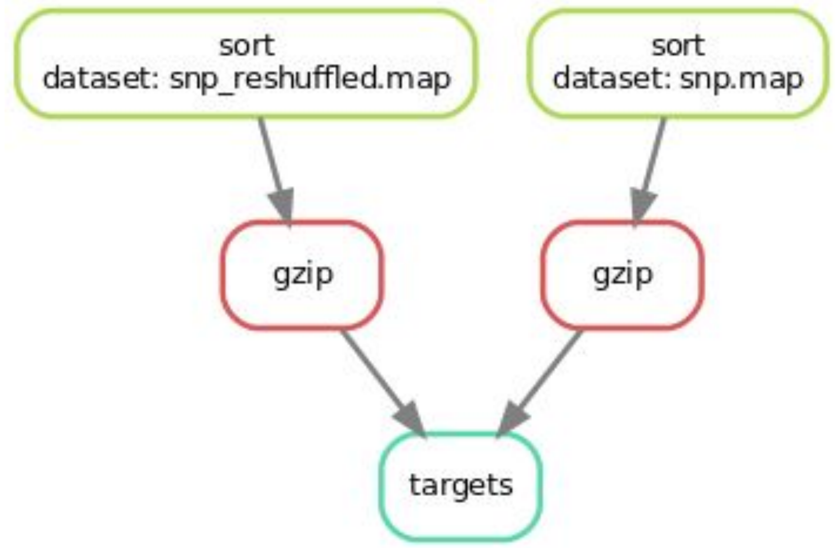
```
snakemake --snakefile example_snakefile -c 1
```

- actual run
- check output
- what happens if you unzip the sorted file? (try first the dry run, then the actual run)



Snakemake - multiple samples

```
## samples
DATASETS = ['snp.map', 'snp_reshuffled.map']


## target rule: files that we want to generate
rule targets:
    input:
        expand("{dataset}.sorted.gz", dataset=DATASETS)


## step 2 - rule to compress the sorted map file
rule gzip:
    input:
        "{dataset}.sorted"
    output:
        "{dataset}.sorted.gz"
    shell:
        "gzip {input}"


## step 1 - rule to sort the map file
rule sort:
    input:
        "{dataset}"
    output:
        "{dataset}.sorted"
    shell:
        "sort -n -k1 {input} > {output}"
```



Snakemake - multiple samples

```
snakemake -n --snakefile example_snakefile_2 | dot -Tsvg >
dag_example.svg

snakemake --snakefile example_snakefile_2 --cores 1
```

- dry run
- actual run

The GWAS pipeline

GWAS pipeline - the atomized steps

1. download and prepare the data
2. data preprocessing and filtering
3. imputation of missing genotypes
4. GWAS

GWAS pipeline - the atomized steps

1. download and prepare the data
2. data preprocessing and filtering
3. imputation of missing genotypes
4. GWAS

- each folder contains a bash script to remove intermediate and output files (reset the step): **clean_folder.sh**
- set the correct paths to resources in the bash scripts (e.g. **Plink**, **Beagle** etc.)

bioinformatic pipelines

The GWAS pipeline for rice plant height

- Prepare the environment:

```
mkdir data
mkdir steps
cp -r ../software .
cp ../4.gwas/gwas_rrblup.R software
cp ../4.gwas/gwas_statgengwas.R software
cp ../1.preparatory_steps/prep_rice_data_pipeline.R software
cp ../cross_reference/rice_group.reference software
```

GWAS pipeline - from scripts to steps

1. download and prepare the data
    a. **1.get_data.sh → get_rice_data.sh, get_dog_data.sh**
2. data preprocessing and filtering
    a. 01_data_handling_and_eda.sh [optional]
    b. 2.steps_filtering.sh
3. imputation of missing genotypes
    a. 3.step_imputation.sh
4. GWAS
    a. 4.gwas.sh

GWAS pipeline - from scripts to steps

1. download and prepare the data
    a. 1.get_data.sh → get_rice_data.sh, get_dog_data.sh
2. data preprocessing and filtering
    a. 01_data_handling_and_eda.sh [optional]
    b. **2.steps_filtering.sh → rule filter_genotypes**

```
plink --ped rice.ped --map rice.map --geno 0.05 --mind 0.2 --maf
0.05 --recode --out rice_filtered
```

3. imputation of missing genotypes
    a. 3.step_imputation.sh
4. GWAS
    a. 4.gwas.sh

GWAS pipeline - from scripts to steps

1. download and prepare the data
   a. 1.get_data.sh → get_rice_data.sh, get_dog_data.sh
2. data preprocessing and filtering
   a. 01_data_handling_and_eda.sh [optional]
   b. 2.steps_filtering.sh → rule filter_genotypes
3. imputation of missing genotypes
   a. **3.imputation.sh → rule impute_genotypes**

```
beagle gt=rice_filtered.vcf out=rice_imputed
```

4. GWAS
   a. 4.gwas.sh

GWAS pipeline - from scripts to steps

1. download and prepare the data
   a. 1.get_data.sh → get_rice_data.sh, get_dog_data.sh
2. data preprocessing and filtering
   a. 01_data_handling_and_eda.sh [optional]
   b. 2.steps_filtering.sh → rule filter_genotypes
3. imputation of missing genotypes
   a. 3.step_imputation.sh → rule impute_genotypes
4. GWAS
   a. **4.gwas.sh → rule gwas_kinship**

```
Rscript --vanilla gwas_rrblup.R genotype_file=rice_imputed.raw
snp_map=rice_imputed.map phenotype_file=rice_phenotypes.txt trait=PH
trait_label=plant_height
```

bioinformatic pipelines

Let’s assemble our GWAS pipeline with Snakemake!

- Begin with the rice dataset (continuous phenotype)

bioinformatic pipelines

The GWAS pipeline for rice plant height

- The Snakemake pipeline file: **Snakefile_GWAS.continuous**
    1. get_data
    2. filter_genotypes
    3. ped2vcf
    4. impute_genotypes
    5. vcf2ped
    6. plink_recodeA
    7. gwas_kinship

bioinformatic pipelines

The GWAS pipeline for rice plant height

- The Snakemake pipeline file: **Snakefile_GWAS.continuous**

```
snakemake -n -p -s Snakefile_GWAS.continuous
snakemake --dag -n -s Snakefile_GWAS.continuous | dot -Tsvg > dag_rice.svg
```

bioinformatic pipelines

The GWAS pipeline for rice plant height

- The Snakemake pipeline file: **Snakefile_GWAS.continuous**

```
snakemake -n -p -s Snakefile_GWAS.continuous
snakemake --dag -n -s Snakefile_GWAS.continuous | dot -Tsvg > dag_rice.svg
snakemake -s Snakefile_GWAS.continuous --cores 1
```

bioinformatic pipelines

The GWAS pipeline for dogs cleft lip (binary phenotype)

- The Snakemake pipeline file: **Snakefile_GWAS.binary**
- include preparatory steps in a bash script

```
bash snakefile_command_line_binary.sh
```

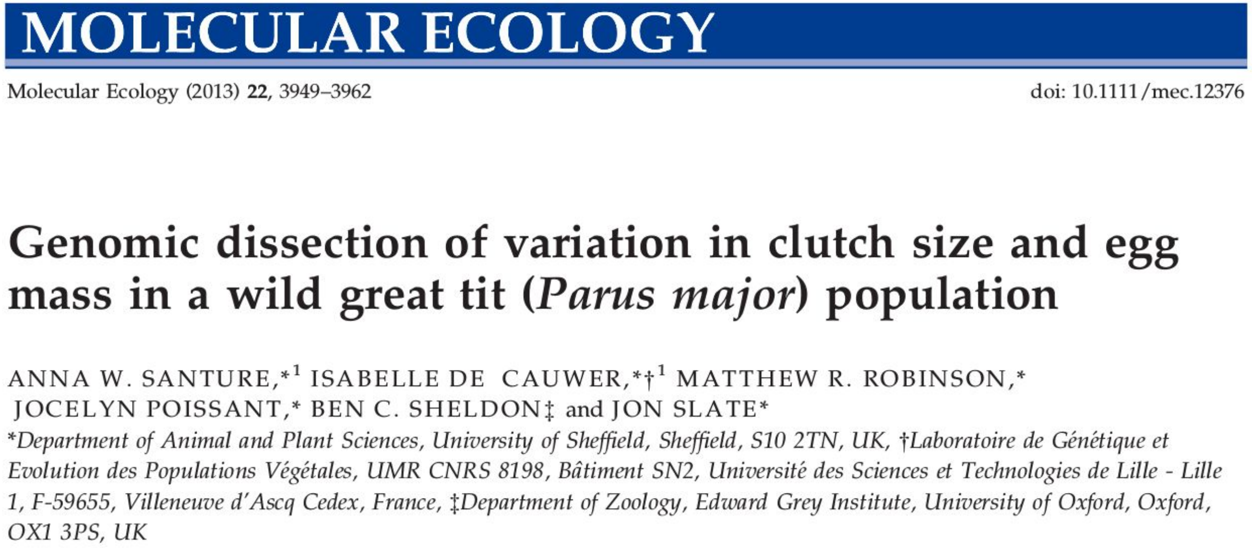
bioinformatic pipelines

The GWAS pipeline for dogs cleft lip (binary phenotype)

- The Snakemake pipeline file: **Snakefile_GWAS.binary**
- include preparatory steps in a bash script

```
bash snakefile_command_line_binary.sh
bash snakefile_command_line_knni.sh
```

GWAS pipeline - assignment



MOLECULAR ECOLOGY

Molecular Ecology (2013) **22**, 3949–3962 doi: 10.1111/mec.12376

Genomic dissection of variation in clutch size and egg mass in a wild great tit (*Parus major*) population

ANNA W. SANTURE,*[1] ISABELLE DE CAUWER,*†[1] MATTHEW R. ROBINSON,* JOCELYN POISSANT,* BEN C. SHELDON‡ and JON SLATE*

**Department of Animal and Plant Sciences, University of Sheffield, Sheffield, S10 2TN, UK, †Laboratoire de Génétique et Evolution des Populations Végétales, UMR CNRS 8198, Bâtiment SN2, Université des Sciences et Technologies de Lille - Lille 1, F-59655, Villeneuve d'Ascq Cedex, France, ‡Department of Zoology, Edward Grey Institute, University of Oxford, Oxford, OX1 3PS, UK*

data from a paper on genetic analysis of **clutch size** and **egg mass** in *Parus major*

GWAS pipeline - assignment

<u>Phenotypes</u>

- egg numbers (clutch size)
- egg mass

GWAS pipeline - assignment

- repository: https://datadryad.org/resource/doi:10.5061/dryad.ck1rq
- article: https://onlinelibrary.wiley.com/doi/pdf/10.1111/mec.12376

focus on:

1. understand the data and the problem/project at hand
2. manipulate the data to get them in the same format as the dogs and rice data before the filtering/imputation steps

challenges:

- multiple phenotypes per individual (over time)
- errors/missing values in the genotype data

GWAS pipeline - assignment

Alternatively, you could use another phenotype from the rice dataset.

File plantgrainPhenotypes.txt

- Panicle length (PL)
- Length of the flag leaf (FLL)
- Width of the flag leaf (FLW)
- Length of the seed (SL)
- Seed width (SW)
- Seed length/width ratio (SR)

you can also consider "binarising" a trait

GWAS pipeline - assignment

1. Get and prepare the data
2. Explore and filter the data
3. Impute missing genotypes
4. GWAS
5. Make a R Markdown report or slides with steps, results and interpretation

optional

- add extra steps: e.g. get genes from SNPs
- make a step in the pipeline to prepare the environment (folder to make, files to copy etc.)

GWAS pipeline - collaborative exercise

- Build your own pipeline!
  a. Download the data
  b. Prepare the data (look at the phenotypes!)
  c. Filter the data
  d. Impute missing genotypes
  e. Run the GWAS
- Data on stump tail sperm defect of Swiss Large White boars (https://zenodo.org/record/4081475#.YKPfmnUzZhE)

1. Try on your own (individually, groups)
2. Let's do it all together!

NEXT LECTURE

Introduction to GWAS: collaborative exercise