

Programming and Data Science for Biology (PDSB)

Session 12
Spring 2018

Today's topics

1. Conda environments Py2/3
2. Parallelization
3. RAD-seq, ipyrad, and the API
4. Genomic analyses with API
wrappers for parallel execution,
and writing wrappers.

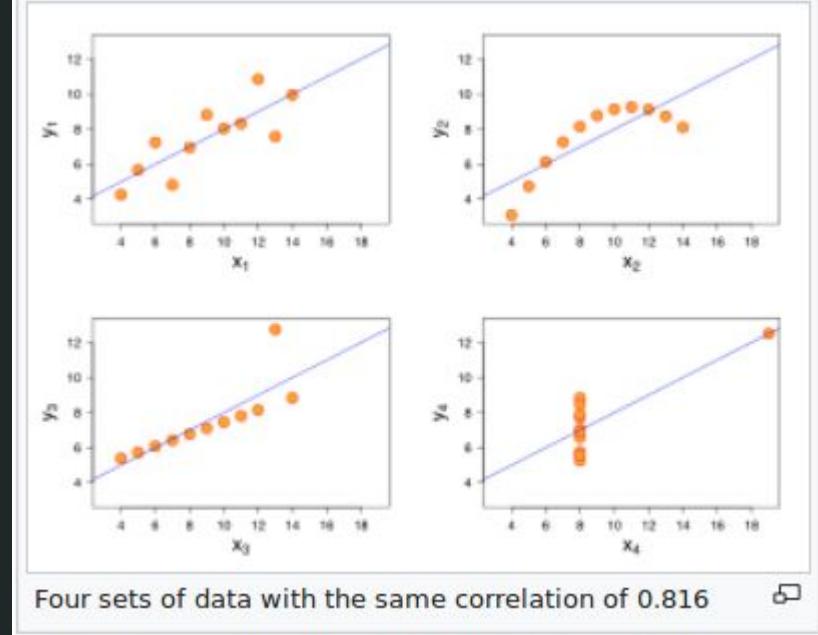
Upcoming topics

1. **Today:** Parallel code,
presentation example, and
genomics part I.
 2. **Next week:** Presentations part I.
And genomics part II.
 3. **Two weeks:** Presentations part II.
And genomics part III.
-

Last week's assignment:
Plotting tools.

Graph types

Plotting helps to visualize data, and thus avoid errors and motivate hypotheses to test.



Wikipedia

Conda environments:

Isolating tools for when different requirements are needed, or for temporary use.

X - notebooks/ - Google Chrome

Google S... X PDSB-Lect... X PDSB-Lect... X multithrea... X notebook X ipyrad-doc... X Future We... X Deren

localhost:9317/tree/notebooks

jupyter Logout

Files Running Clusters

Select items to perform actions on them.

Upload New

	Name	Last Modified
<input type="checkbox"/>	0	seconds ago
	..	seconds ago
<input type="checkbox"/>	analysis-bpp	5 months ago
<input type="checkbox"/>	ipsimdata	9 months ago
<input type="checkbox"/>	development.ipynb	5 months ago
<input type="checkbox"/>	ipyrad-doc.ipynb	Running a minute ago
<input type="checkbox"/>	scratch.ipynb	10 hours ago
<input type="checkbox"/>	Untitled.ipynb	10 hours ago
<input type="checkbox"/>	ipyrad_log.txt	9 months ago
<input type="checkbox"/>	test1.json	9 months ago

Conda environments

Managing conda environments:

<https://conda.io/docs/user-guide/tasks/manage-environments.html>

Conda environments allow you to keep different versions of your software separate.

The main thing you likely want is to have a py2 and py3 environment. Using ipykernel you can start a notebook and access both.

```
# set conda in your path (if not already there)
export PATH=~/miniconda3/bin/:$PATH
```

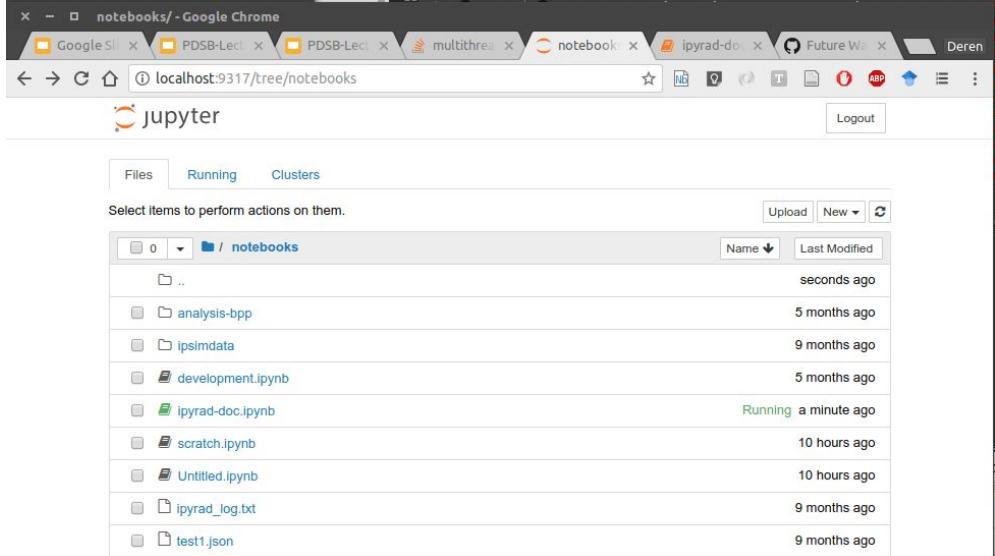
```
# create a py27 env and register kernel
conda create -n py27 python=2.7
source activate py27
conda install notebook ipykernel
ipython kernel install --user
```

```
# create a py36 env and register kernel
conda create -n py36 python=3.6
source activate py36
conda install notebook ipykernel
ipython kernel install --user
```

```
# show your conda environments
conda env list
```

Conda environments

Access multiple **kernels** from jupyter. You can additionally install kernels to access other languages like *bash*, *R*, and *julia*.



Presentations

An example and guide

Project presentations

Main Points --

1. Describe your proposal.
2. Demonstrate your progress.

Topics to include:

1. How are you are representing your data (e.g., as an array, a network, a dictionary).
2. Where are you stuck or unsure?
3. What do you plan to explore next to solve the problem?

The screenshot shows a GitHub repository page for 'records'. The URL is https://github.com/programming-for-bio/records. The repository is described as 'An example group project for PDSB'. It has 15 commits, 2 branches, 0 releases, and 1 contributor. The latest commit was made 22 days ago. The repository includes files like data, notebooks, records, .gitignore, LICENSE, README.md, and setup.py. A file named records is currently selected. The GitHub interface includes a header with navigation links like Pull requests, Issues, Marketplace, and Explore, and a sidebar with options like Watch, Star, Fork, and Clone or download.

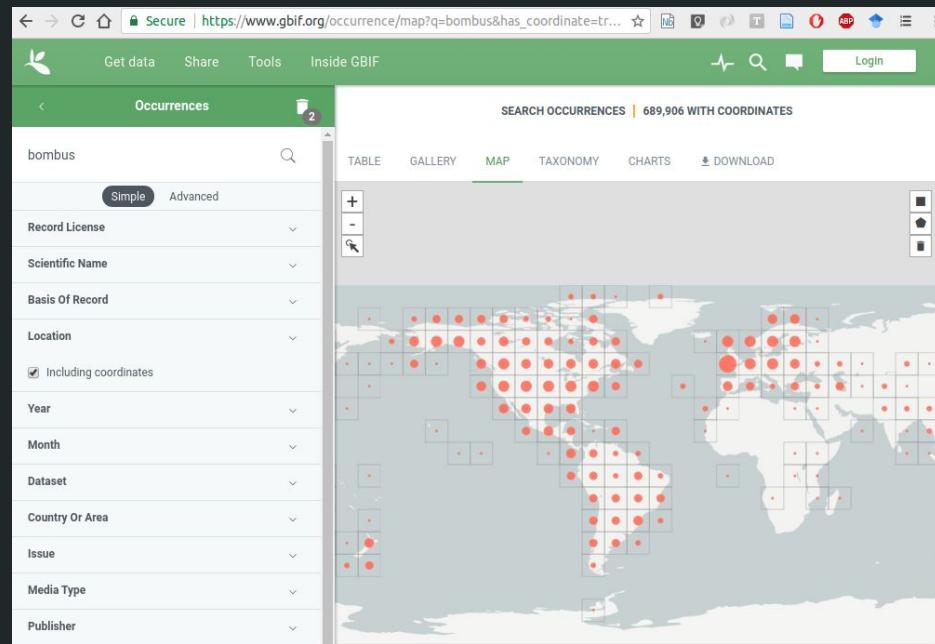
File	Description	Time Ago
bombus big sdf data added	eaton-lab	Latest commit cd2ab51 22 days ago
data		22 days ago
notebooks		22 days ago
records		22 days ago
.gitignore		23 days ago
LICENSE		23 days ago
README.md	Initial commit	23 days ago
setup.py	added dependencies to readme	22 days ago
	parse version in init	22 days ago

The records library

The problem: extracting and analyzing spatial data from GBIF using the API is not easy, I'd like to develop a simple tool for doing this. I want to ask, has bumblebee diversity changed through time?

The data: GBIF occurrence records, in particular my example dataset is of *Bombus* (bumblebees).

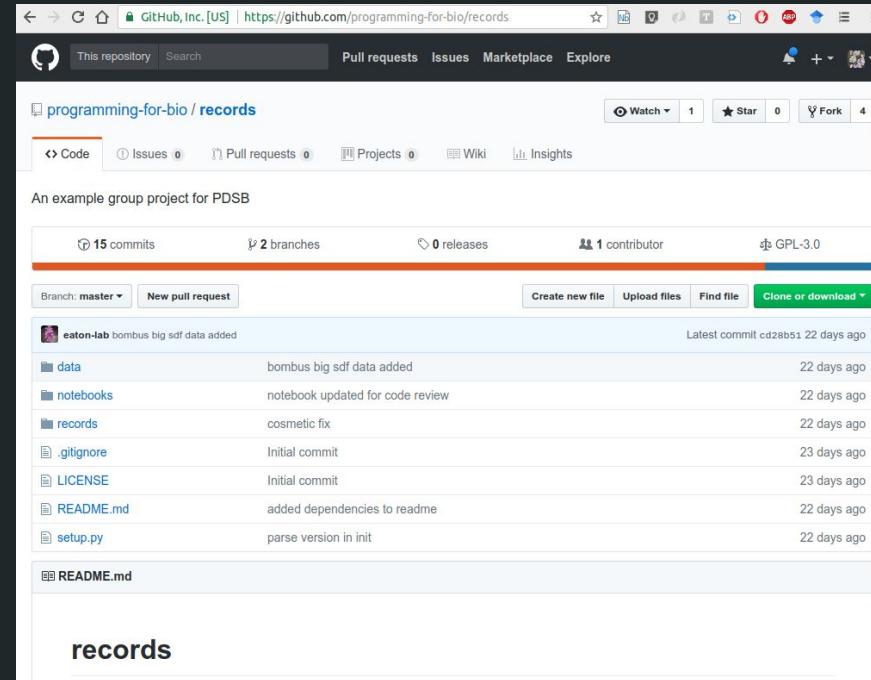
The tools: I am using the `requests` package to access data from GBIF using the REST API. Then I'm using `pandas` to organize it as a dataframe, filter and clean the data, and finally `toyplot` for plotting and `pymc3` for statistical analyses.



The records library

The novelty: By combining data extraction and analysis into a single pipeline my tool can be reused on many other types of data sets. The statistical method I will apply is also novel in that I have not seen it applied to GBIF data before. And this question about Bombus diversity is relevant to conservation.

The goal: A Python library with an easy to use API that can be easily imported and used by other users. I will have an example notebook demonstrating its use with plots and statistics as well for the Bombus example.



The **records** library

How am I developing it?

I've been writing code in sublimetext and testing it in a jupyter notebook.

I've written two main Class objects, one is the **Records** class, used to make queries to GBIF. The other is the **Epochs** Class, used to organize Records results into a dataframe and provide functions for users to calculate statistics.

```
import records
import pandas as pd
import pymc3 as pm
import toyplot
```

Query database to get records of Bombus

```
# retrieve records from GBIF database
rec = records.EPOCHS("Bombus", 1900, 2015, 5)
```

```
# save dataframe to disk
rec.df.to_csv("/home/deren/PDSB/records/data/Bombus-1900-2015.csv")
```

The records library

See the code on GitHub:

<http://github.com/programming-for-bio/records/>

```
class Records:
    """
    Returns a Records class instance with GBIF occurrence records stored
    in a pandas DataFrame for a queried taxon between a range of years.

    Parameters:
    -----
    q: str
        Query taxonomic name.
    interval: tuple
        Range of years to return results for. Should be (min, max) tuple.

    Attributes:
    -----
    baseurl: The REST API URL for GBIF.org.
    params: The parameter dictionary to filter GBIF search.
    df: Pandas DataFrame with returned records.
    sdf: A view of the 'df' DataFrame selecting only three relevant columns.
    """
    def __init__(self, q, interval, **kwargs):
        # the API url for searching GBIF occurrences
        self.baseurl = "http://api.gbif.org/v1/occurrence/search?"

        # the default REST API options plus user entered args
        self.params = {
            'q': q,
            'year': ",".join([str(i) for i in interval]),
            'basisOfRecord': "PRESERVED_SPECIMEN",
            'hasCoordinate': "true",
            'hasGeospatialIssue': "false",
            'country': "US",
            "offset": "0",
            "limit": "300",
        }

        # allow users to enter or modify other params using kwargs
        self.params.update(kwargs)

        # run the request query until all records are obtained
        self.df = pd.DataFrame(self._get_all_records())
```

The **records** library

The .df and .sdf attributes return a full view and shortened view of the dataframe, respectively.

Access dataframe of results

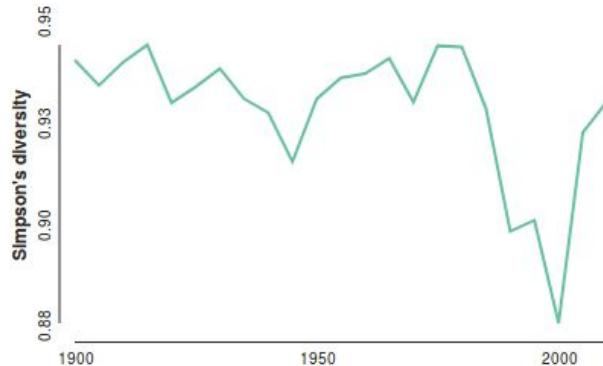
```
rec.sdf.head(10)
```

	species	year	epoch	country	stateProvince
0	Bombus fervidus	1900	1900	United States	Kansas
1	Bombus fervidus	1900	1900	United States	Kansas
2	Bombus vosnesenskii	1900	1900	United States	California
3	Bombus affinis	1900	1900	United States	New York
4	Bombus fervidus	1900	1900	United States	Kansas
5	Bombus pensylvanicus	1900	1900	United States	Nebraska
6	Bombus fervidus	1900	1900	United States	Kansas
7	Bombus variabilis	1900	1900	United States	Illinois
8	Bombus fervidus	1900	1900	United States	Kansas
9	Bombus sylvicola	1900	1900	United States	Alaska

The records library

The .df and .sdf attributes return a full view and shortened view of the dataframe, respectively.

From Epochs class objects users can calculate Simpson's diversity (probability two individuals are the same species).



Calculate diversity indices

```
# diversity across all state by epoch  
rec.simpsons_diversity("epoch")
```

epoch	species
1900	0.940813
1905	0.934551
1910	0.940228
1915	0.944503
1920	0.930227
1925	0.934184
1930	0.938666
1935	0.931209
1940	0.927794
1945	0.915737
1950	0.931189
1955	0.936433
1960	0.937387
1965	0.941193
1970	0.930395
1975	0.944311
1980	0.944033
1985	0.928776
1990	0.898571
1995	0.901256
2000	0.875938
2005	0.922909
2010	0.930507

Name: species, dtype: float64

The records library

What's next?

- Groupby state and epoch
- Filter bad state names
- Apply switchpoint model in pymc3 for each state using a hierarchical model.

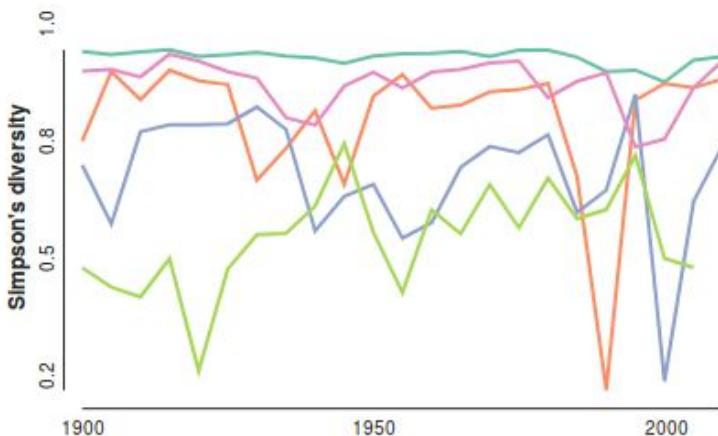
```
sdf = df.unstack()
mask = (
    (sdf.index == '[California]') |
    (sdf.index == '[Not Stated]') |
    (sdf.index == 'Not Stated') |
    (sdf.index == 'Colorado - New Mexico - Oklahoma') |
    (sdf.index == 'District Of Columbia') |
    (sdf.index == 'Washington, D. C.') |
    (sdf.index == 'Mass') |
    (sdf.index == 'District of Columbia')
)
sdf = sdf[~mask].T

# state with more than 5 NaNs are removed
mask = np.sum(sdf.isna(), axis=0) <= 5
sdf.loc[:, mask].head()
```

stateProvince	Alaska	Arizona	California	Colorado	Connecticut	Florida	Georgia	Idaho	
epoch									
1900	0.750000	0.584933	0.699000	0.899221	0.743802	0.480000	0.565089	0.81632	
1905	0.898438	0.145957	0.573519	0.902970	0.507545	0.438776	0.269231	0.75510	
1910	0.838296	NaN	0.769859	0.886856	0.574827	0.417778	0.675900	0.76367	
1915	0.901213	0.909970	0.785077	0.935441	0.757847	0.499241	0.691358	0.84760	
1920	0.878903	NaN	0.784623	0.920467	0.331066	0.259969	0.647500	0.85937	

The **records** library

There is a lot of variability by state, so my analyses should probably take this into account.



```
# diversity across all state by epoch
c, a, m = toyplot.plot(
    div.epoch, div.species,
    width=450, height=300,
    ylabel="Simpson's diversity");

a.plot(sdf.index, sdf.Alaska)
a.plot(sdf.index, sdf.California)
a.plot(sdf.index, sdf.Colorado)
a.plot(sdf.index, sdf.Florida)

<toyplot.mark.Plot at 0x7f2e2b197828>
```

The records library

What's next?

- Groupby state and epoch
- Filter bad state names
- Apply switchpoint model in pymc3 for each state using a hierarchical model.

The switchpoint model is still in progress.

I'm trying to figure out how to apply a hierarchical model and switchpoint models together. I'm reading more on the pymc3 docs and running tests to figure it out.

```
with pm.Model() as model:  
  
    # sample switchpoint from within the bounds of our data timescale  
    switchpoint = pm.DiscreteUniform(  
        'switchpoint',  
        lower=div.epoch.min(),  
        upper=div.epoch.max(),  
        testval=1980)  
  
    # Priors for pre- and post-switch rates for number of disasters  
    early_rate = pm.Exponential('early_rate', 1.)  
    late_rate = pm.Exponential('late_rate', 1.)  
  
    # Allocate appropriate Poisson rates to years before and after current  
    rate = pm.math.switch(switchpoint >= div.epoch, early_rate, late_rate)  
    diversity = pm.Poisson('disasters', rate, observed=div.species)
```

```
with model:  
    trace = pm.sample(10000)
```

```
Assigned Metropolis to switchpoint  
Assigned NUTS to early_rate_log__  
Assigned NUTS to late_rate_log__  
90%|██████████| 9473/10500 [00:08<00:00, 1177.62it/s]
```

The records library

Questions, recommendations, ideas?

```
with pm.Model() as model:  
  
    # sample switchpoint from within the bounds of our data timescale  
    switchpoint = pm.DiscreteUniform(  
        'switchpoint',  
        lower=div.epoch.min(),  
        upper=div.epoch.max(),  
        testval=1980)  
  
    # Priors for pre- and post-switch rates for number of disasters  
    early_rate = pm.Exponential('early_rate', 1.)  
    late_rate = pm.Exponential('late_rate', 1.)  
  
    # Allocate appropriate Poisson rates to years before and after current  
    rate = pm.math.switch(switchpoint >= div.epoch, early_rate, late_rate)  
    diversity = pm.Poisson('disasters', rate, observed=div.species)  
  
with model:  
    trace = pm.sample(10000)  
  
Assigned Metropolis to switchpoint  
Assigned NUTS to early_rate_log__  
Assigned NUTS to late_rate_log__  
90%|██████████| 9473/10500 [00:08<00:00, 1177.62it/s]
```

Parallelization

Using it and writing code for it

Running parallel code

Ideally, running parallel code should be easy, as simple as designating the number of processors you want your analysis to use.

Example, in pymc3 we could provide `njobs` as an argument to run mcmc chains on separate processors in parallel. Easy!

However, knowing more about the architecture of parallelization can help us to use it more efficiently.

```
with model:  
    trace2 = pm.sample(njobs=4, chain=4)  
  
Auto-assigning NUTS sampler...  
Initializing NUTS using jitter+adapt_diag...  
18%|██████████| 185/1000 [00:33<02:26, 5.58it/s]
```

The Global Interpreter Lock (GIL) in Python

If you have multiple functions running at the same time, then one might change the state of an object in memory when another one is trying to access it, and this could cause problems.

To avoid this, Python **locks** processes from operating on the same memory space. This is great and allows you to write simple code without worrying about complicated problems. However, to write parallel code we then need to *work around the GIL*.

```
# run one function
res1 = func1()

# progress is 'blocked' until func1 finishes,
# then the next code block can run.
res2 = func2()
```

Forms of parallel code

Concurrency is when two or more tasks can start, run, and complete in overlapping time periods. It doesn't necessarily mean they'll ever both be running at the same instant. Eg. multitasking on a single-core machine.

Parallelism is when two or more tasks are executed simultaneously.

Forms of parallel code

A **thread** is a sequence of instructions within a process. It can be thought of as a lightweight process. Threads share the same memory space.

A **process** is an instance of a program running in a computer which can contain one or more threads. A process has its independent memory space.

In Py3 concurrency is easier to use, and is often used in place of multithreading.

Concurrency

Here the *memory-space* is shared between all running functions. The code is mostly running in parallel, but it is not using more than one processor.

This is best used for functions that take time to complete, but are not doing much work during that time. This would include I/O limited tasks, or network limited (e.g., web requests).

[Notebooks/nb-12.1-parallel-threading.ipynb](#)

```
with ThreadPoolExecutor(max_workers=4) as executor:  
    # submit jobs to threads  
    jobs = [executor.submit(gsearch, q) for q in queries]  
  
    # collect results  
    results = [i.result() for i in jobs]
```

Multi-processing

The general workflow is to open a **client** (pool, executor) that can distribute jobs to workers.

Then send jobs to workers (submit, apply_async).

The collect the results from the jobs once they finish.

Notebooks/nb-12.2-multiprocess-ipyparallel.ipynb

```
with ProcessPoolExecutor(max_workers=4) as executor:  
  
    # submit queries to threads  
    jobs = [executor.submit(gsearch, q) for q in queries]  
  
    # collect results  
    results = [i.result() for i in jobs]
```

```
with mp.Pool(processes=4) as pool:  
  
    # submit queries to threads (with args as tuples)  
    jobs = [pool.apply_async(who, (1,)) for q in queries]  
  
    # collect results  
    results = [i.get() for i in jobs]
```

Blocking vs. Async

The normal way in which we execute code is that when we execute a function call our progress is **blocked** until that function finishes.

Asynchronous execution allows us to send a job to run remotely and to continue working locally, to send more jobs or to query whether the remote job is finished.

Notebooks/nb-12.2-multiprocess-ipyparallel.ipynk

```
# start a pool
with mp.Pool(processes=4) as pool:

    # submit a job that will run successfully
    async0 = pool.apply_async(who, [2])

    # submit a job that will fail
    async1 = pool.apply_async(who, ['apple'])

    # join jobs into an iterable
    jobs = [async0, async1]

    # iterate over async results and print result of successful jobs
    while 1:
        for job in jobs:
            # skip over the job for now if not ready yet
            if job.ready():
                if job.successful():
                    print('{} was successful'.format(job))
                else:
                    print('{} had an error'.format(job))
            # remove job from queue
            jobs.remove(job)

    # end loop if all jobs are finished
    time.sleep(0.5)
    if len(jobs) == 0:
        break
```

```
<multiprocessing.pool.ApplyResult object at 0x7f7a58dcedd8> had an error
<multiprocessing.pool.ApplyResult object at 0x7f7a58dce9e8> was successful
```

Genomic assembly and analysis with ipyrad: And a dive into the ipyrad API

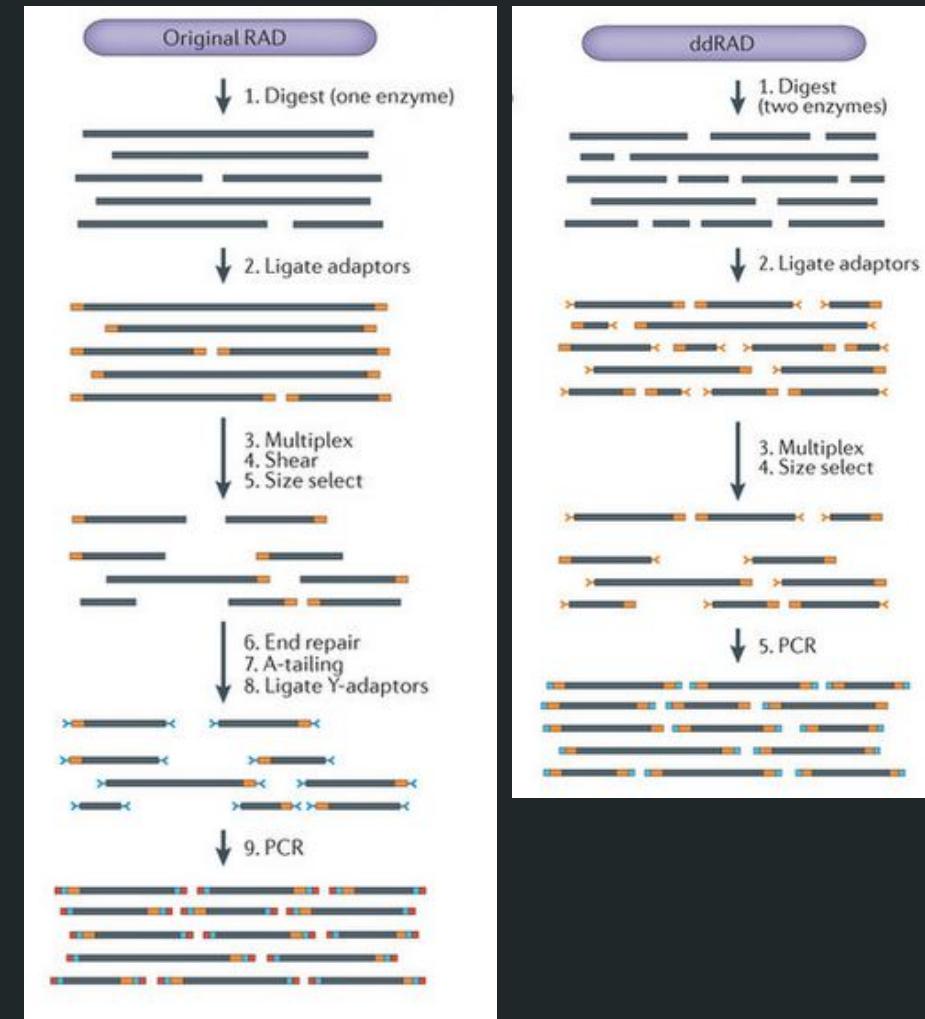
RAD-seq genomics

Restriction-site associated DNA is a type of genomic data that samples the genome at places where certain recognition sites occur.

Review

Harnessing the power of RADseq for ecological and evolutionary genomics

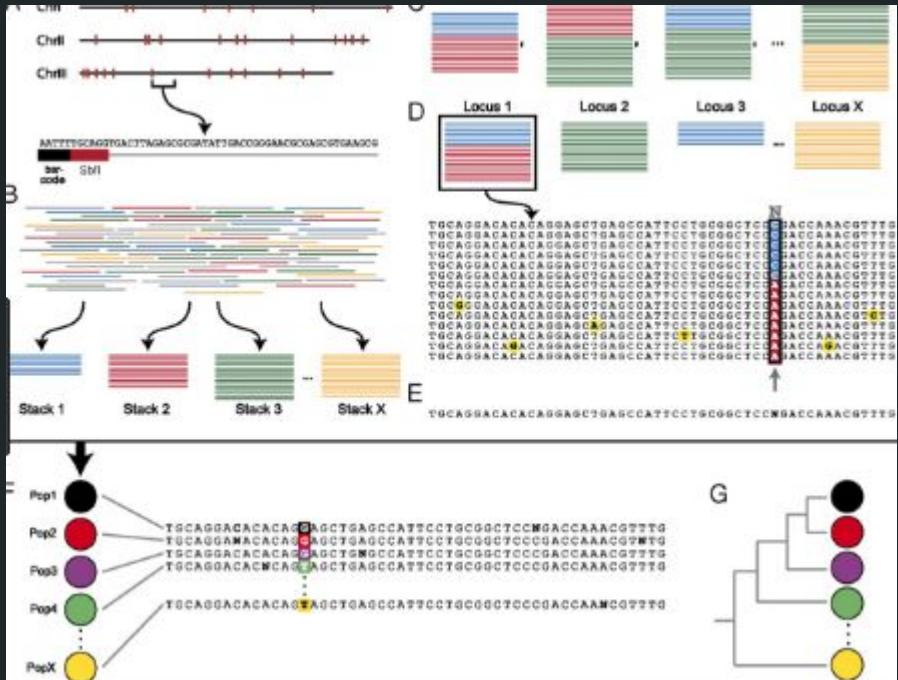
Kimberly R. Andrews , Jeffrey M. Good, Michael R. Miller, Gordon Luikart & Paula A. Hohenlohe



RAD-seq genomics

It is a convenient method for sampling thousands of loci from hundreds or thousands of individuals.

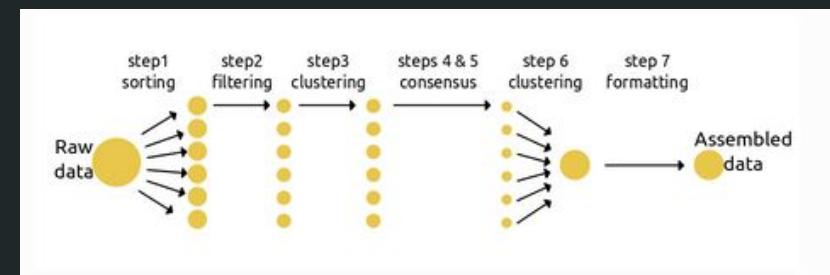
Short-read sequencing technologies yield reads that are ~50-150 bp long, which we need to *assemble* bioinformatically to identify which reads are from the same genomic region, to correct for base-calling errors, and to identify homologous sequences across samples.



RAD-seq genomics

One program for this is *ipyrad*, which I've developed over the last 5 years, and with the help of a collaborator have now expanded to include complex assembly options and analysis tools.

Ipyrad uses *ipyparallel* to distribute jobs over arbitrarily large computing clusters while also allowing users to work interactively and perform their analysis in reproducible jupyter notebooks.



ipyrad
latest

Search docs

The ipyrad Ethos
Features
Installation
Input data files
Assembly Steps and Branching
Assembly Parameters
Assembly Methods
Tutorials (running ipyrad)
Jupyter and the ipyrad API
The ipyrad analysis toolkit
Release Notes
Support

Read the Docs v: latest ▾

Docs » ipyrad: interactive assembly and analysis of RADseq data sets [Edit on GitHub](#)

ipyrad: interactive assembly and analysis of RADseq data sets

Welcome to ipyrad, an interactive toolkit for assembly and analysis of restriction-site associated genomic data sets (e.g., RAD, ddRAD, GBS) for population genetic and phylogenetic studies.

In this documentation you can learn about the various features of ipyrad ([Features](#)), the broader motivation behind its design ([Ethos](#)), and access detailed [Tutorials](#). Several new Analysis tools are included with ipyrad, and we are also building a large Cookbook with code for many common downstream analyses.

Got questions about ipyrad? Join the conversation at [gitter](#).

Documentation

- The ipyrad Ethos
- Features
- Installation
- Input data files
- Assembly Steps and Branching
- Assembly Parameters
- Assembly Methods
- Tutorials (running ipyrad)
- Jupyter and the ipyrad API
- The ipyrad analysis toolkit

RAD-seq genomics

Installation: ipyrad is written for Python 2.7

<http://ipyrad.readthedocs.io/installation.html>

Install a python2 conda environment and install with a simple conda command. This pulls in all dependencies, of which there are many, and allows users to easily install the software locally.

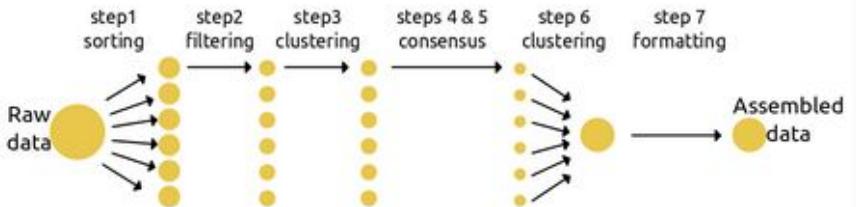
```
# install ipyrad from conda  
conda install ipyrad -c ipyrad
```

RAD-seq genomics

Step 1:

Sorting/Loading reads.

Load data from .fastq or .fastq.gz files. If the reads are not sorted into samples yet then you can provide a barcodes file with index information to demultiplex reads.



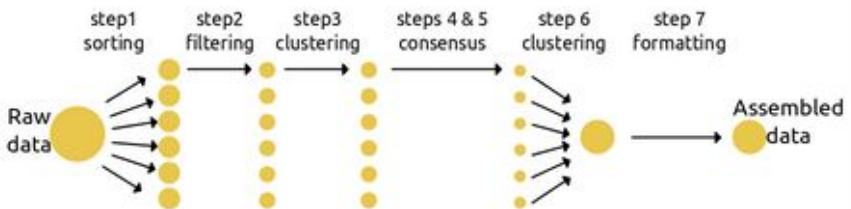
```
less /rigel/edu/w4050/files/example_empirical_rad/29154_superba.fastq.gz
```

RAD-seq genomics

Step 2:

Filter reads

Trim illumina adapter sequences and low quality bases from the ends of reads.



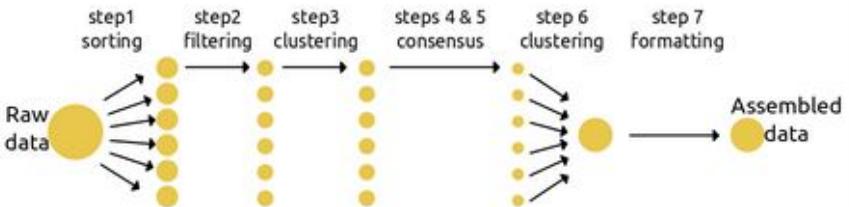
```
less /rigel/edu/w4050/files/analysis-ipyrad/Pedicularis_edits/\\n29154 superba.trimmed R1 .fastq.gz
```

RAD-seq genomics

Step 3:

Cluster/map reads

Identify reads that are from the same RAD locus within each sample and cluster them together by sequence similarity (denovo) or by mapping to a reference genome (reference).



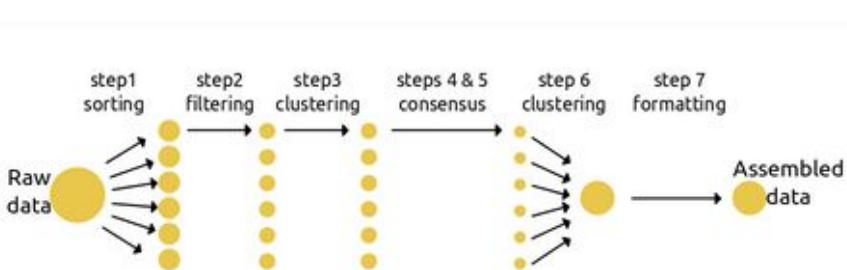
```
less w4050/files/analysis-ipyrad/Pedicularis_clust_0.9/\n32082_przewalskii.clustS.gz
```

```
0001b983dbc693d31d9d971be660771a;size=8;*\nTCAGGAGAAATCATGCAGCATCAGATGCCATGGCATTATCATTACAACAGCATGCAAATTAAATAGAATT\n0f09d01ef88543a687f35db247a1c0f;size=8;+\nTCAGGAGAAATCATGCAGCATGAGATGCCATGGCATTATCATTACAACACCATGCAAATTAAATAGAATT\nbb012c69ef74409a44b5581ffa8dc260;size=1;+\nTCAGGAGAAATCATGCCATGAGATGCCATGGCATTATCATTACAACACCATGCAAATTAAATAGAATT\n9f97cb575f6b85b9ed83dc2dacf10aec;size=1;+\nTCAGGAGAAATCATGCAGCATGAGATGCCATGGCATTATCATT-----\n//\n//\n00024f7687a7bb684a2c2397ec2a0d0d;size=14;*\nTCAGCTCTTCACTCTCGAACGCTCGAACGCCGTTCAAGTCCCTGTTCAACAAATCTGGCGATGATGA\n1bbedead1d4ba95aa97184863d69423c;size=4;+\nTCAGCTCTTCAATCTCGAACGCTCGAACGCCGTTCAAGTCCCTGTTCAACAAATCTGGCGATGATGA\n7c95ba9bb40f3234d910fbc51ded6cc7;size=1;+\nTCAGCTCTTCACTCTCGAACGCTCGAACGCCGTTCAAGTCCCTGTTCAACAAATCTGGCG-----\na9196acfb3f2758dde0d826c8e040b2f;size=1;+\nTCAGCTCTTCACTCTCGAACGCTCGAACGCCGTTCAAGTCCCTGTTCAACAA-----\n5021dd9126bb4b8b608071337a174c32;size=1;+\nTCAGCTCTTCAATCTCGAACGCTCGAACGCCGTTCAAGTCCCTGTTCAACAAATCTGGCGATGATG-\n//\n//\n0003b73fa2fd642a64a64bac96eff0fa;size=599;*\nTCAGCTTGATTATCGTAGAATAAGAGGGGCCGCTTAGGAAATGACTTAGTTAACTAAAAGACAGATGCC\n5ce0ab38773024648ffffd091c903fd37;size=3;+\nTCAGCTTGATTATCGTAGAATAAGAGGGGCCGCTTAGGAAATGACTTAGTTAACTAAAAGACAGATGCC\na2aaa64f49fb7fb579cb9a54cc068d3;size=2;+\nTCAGCTTGATTATCGTAGAATAAGAGGGGCCGCTTAGGAAATGACTTAGTTAACTAAAAGACAGATGCC\ne99f8f219794191e541dcbb752f7f519c;size=2;+\nTCAGCTTGATTATCGTAGAATAAGAGGGGCCGCTTAGGAAATGACTTAGTTAACTAAAAGACAGATGCC\n8d439f488031595710170fe4c518f0b;size=2;+\nTCAGCTTGATTATCGTAGAATAAGAGGGGCCGCTTAGGAAATGACTTAGTT-----\ncbd277bbf2e12ae570e46ec56c845003;size=2;+\nTCAGCTTGATTATCGTAGAATAAGAGGGGCCGCTTAGGAAATGACTTAGTTAACT-----\n77bc94ead6597d65a17d5603e468ebea;size=2;+
```

RAD-seq genomics

Steps 4-5:

Estimate parameters and make consensus base calls and haplotype calls.



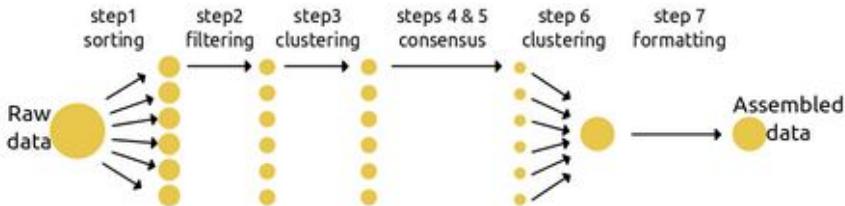
```
less w4050/files/analysis-ipyrad/Pedicularis_consens/29154_superba.consens.gz
```

```
>29154_superba_0
TGCAGGATGATAACAAAGCATTGGACCGGCTCACCAATCCAAGAGGCTGCTACTAGAAAACAGAGAGAACT
>29154_superba_1
TGCAGCTTGATTATCGATAATAAGAGGGGCCGTTAGGAATGACTTAGTTAACTTAAAGACAGATGCC
>29154_superba_2
TGCAGCGGCGCTAGAACGGTGTGAYAATTAAAGGTYTGTCTATTGATTATTCAATTCAAATAGGAACATAA
>29154_superba_3
TGCAGATTGAAATTCCCAGAAAAGATTGTTATCAAYAAACGGAATTGRTGTMTCGTTMCAACAGTAM
>29154_superba_4
TGCAGTAAATTAGTGAAGTTATTGTAATGTTATTCAAGGAAGCAGAGAGGAAATCCAATTATTATGCCT
>29154_superba_5
TGCAGTCGCTGMCYAGAAAAATGATGTTGGTTGATGTTGGCAACATCAGACSCAANATYRATGTCGYTRCTC
>29154_superba_6
TGCAGAACTATAAATTTCATCAACCCATTAACTTATGTTCATTTATTAAAAAAAGAAAATACAAGATA
>29154_superba_7
TGCAGTGCACAAAAAAAKGTCGAAAANAAAYGCTTACCAAAATCCACACyGTAGTGCCTGTTGTGTT
>29154_superba_8
TGCAGCTTGAGGAGACAATCATTGACAAGATGCAAGATGGAGTACTAGTGTGGAGGTTAAGGTGAAGAGG
>29154_superba_9
TGCAGTATTGAAGAGAGCCTGSKCGCTATCGTGTGTAstTAATGCCCTAAGAAGCTTCGATTCCGTCGCG
>29154_superba_10
TGCAGNATTGAGGTTGCGTAAANAAAAATAAGATAACCGAACACYGATGTTGTCAGTGGTTGGCAACACCANTCAC
>29154_superba_11
TGCAGCTGTTGATTGATTGCCAGCAACTCCTCCGCACTCTGGCTCGACACTAACTCCATTCCAATTAA
>29154_superba_12
TGCAGCAATTAAAAGAACACTCGGGTCCAAGGAACAAAGGATATAAAAGTCCATGTGTTGTCATCCTCAGT
>29154_superba_13
TGCAGAGTCCAGACATTACAGAAAACCTGTCATTCTAACAAACACACTTTCGCTTAACGACTTCTACT
>29154_superba_14
TGCAGCATTTCGAACTCAAGTAGATATGGCTCAACCGCAACTCCGGNCACAAATTGTATCCAAGGCCT
>29154_superba_15
TGCAGATTATCAGAACATTGCCATATCGCATGATGACTCGATCTTGAGTAAGGCTCATGATGATCAAGTGA
>29154_superba_16
```

RAD-seq genomics

Steps 6:

Cluster/Map across samples -- identify *homologous* sequences derived from the same genomic locus.



```
less w4050/files/analysis-ipyrad/Pedicularis_across/\_Pedicularis_catclust.gz
```

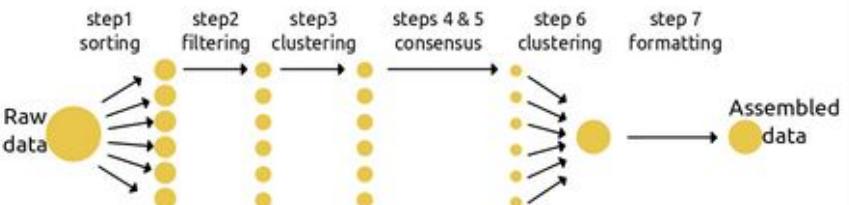
```
29154_superba_1045
TCAGGTGAGACGATCTGCACTTT--TTTTCATCCATAAAAAGAAAAATGGATTT-AAGTCAATATAAATA
30556_thanno_65359
TCAGGTGAGAGGATCTGCACTTT-TTTTTATCCATAAAAAGAAAAATGGATTAAAGTCAATATAAA-
30686_cyathophylla_71328
TCAGGTGAGACGATCTGCACTTTCTTTTATCCATAAAAAGAAAAATGGATTT-AAGTCAATATAAA-
41478_cyathophylloides_61181
TCAGGTGAGACGATCTGCACTTT--TTTTCATCCATAAAAAGAAAAATGGATTAAAGTCAATATAAAT-
41954_cyathophylloides_67315
TCAGGTGAGACGATCTGCACTT--TTTTCATCCATAAAAAGAAAAATGGATTTAAAGTCAATATAAAT-
// 
29154_superba_105
TCAGGCCAGCCCGATCCATCAGGCCCTGTCGGCTTATGCCGGGCTAAATAACCGGATTCYAGCCGGTT
30686_cyathophylla_40275
TCAGGCCAGCCCGATCCATCAGGCCCTGTCGGCTTATCACGGGCAAATAACCGCATTCCAGCCGAAT
// 
// 
29154_superba_107
TCAGGATCATCATTATGTACCGATAATAAGAACCTCACCAATCTGTCATCTTAAATGTCAGAATACT
35236_rex_1103
TCAGGATCATCATTATGTACCGATAATAAGAACCTCACCAATCTCTCATCTTAAATGTCGAATACA
35855_rex_24476
TCAGGATCATCATTATGTACCGATAATAAGAACCTCACCAATCTSTCATCTTAAATGTCAGAATACT
38362_rex_18814
TCAGGATCATCATTATGTACCGATAATAAGAACCTCACCAATCTCTCATCTTAAATGTCAGAATACT
39618_rex_14308
TCAGGATCATCATTATGTACCGATAATAAGAACCTCACCAATCTCTCATCTTAAATGTCAGAATACT
41478_cyathophylloides_359
TCAGGATCATCATTATGTACCGATAATAAGAACCTCACCAATCTGTCATCTTAAATGTCAGAATACT
41954_cyathophylloides_77279
TCAGGATCATCATTATGTACCGATAATAAGAACCTCACCAATCTGTCATCTTAAATGTCAGAATACT
//
```

RAD-seq genomics

Steps 7:

Filter and format output files.

You can apply many different filters at this stage to reduce error-prone regions, or to exclude data with too much missing information.



```
less /rigel/edu/w4050/files/analysis-ipyrad/\min10_outfiles/min10.loci
```

```
29154_superba ATATAACGAGGCCACAGCCGGTTTAAGCCTTACAGACAAAAAAGATGGCCCTTCTTGCATTGGAC  
30556_thamno ATATAATGAGGCCACAGCCGGTTTAAGCCTTACAGACAAAAAAGATGGCCCTTCTTGCATTGGAC  
30686_cyathophylla ATATAACGAGGCCACAGCCGGTTTAAGCCTTACAGACAAAAAAGATGGCCCTTCTTGCATTGGAC  
35236_rex ATATAATGAGGCCACAGCCGGTTTAAGCCTTACAGACAAAAAAGATGGCCCTTCTTGCATTGGAC  
35855_rex ATATAACGAGGCCACAGCCGGTTTAAGCCTTACAGACAAAAAAGATGGCCCTTCTTGCATTGGAC  
38362_rex ATATAATGAGGCCACAGCCGGTTTAAGCCTTACAGACAAAAAAGATGGCCCTTCTTGCATTGGAC  
39618_rex ATATAATGAGGCCACAGCCGGTTTAAGCCTTACAGACAAAAAAGATGGCCCTTCTTGCATTGGAC  
40578_rex ATATAACGAGGCCACAGCCGGTTTAAGCCTTACAGACAAAAAAGATGGCCCTTCTTGCATTGGAC  
41478_cyathophylloides ATATAATGAGGCCACAGCCGGTTTAAGCCTTACAGACAAAAAAGATGGCCCTTCTTGCATTGGAC  
41954_cyathophylloides ATATAATGAGGCCACAGCCGGTTTAAGCCTTACAGACAAAAAAGATGGCCCTTCTTGCATTGGAC  
// * * [17]  
29154_superba CGCGAACGATGAACCATATACTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
30556_thamno CGCGAACGATGAACCATATACTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
30686_cyathophylla CGCGAACGATGAACCATATACTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
32082_przewalskii CGCGAACGATGAGGCC- TTACCTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
33413_thamno CGCGAACGATGAACCATATACTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
35236_rex CGCGAACGATGAACCATATACTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
35855_rex CGCGAACGATGAACCATATACTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
38362_rex CGCGAACGATGAACCATATACTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
40578_rex CGCGAACGATGAACCATATACTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
41478_cyathophylloides CGGGGACGATGAACCATATACTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
41954_cyathophylloides CGGGGACGATGAACCATATACTGATTGCCGTCTCTTCTCTCCGCTCAGCAAGGTCACAAAT  
// * * [17]  
29154_superba AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
30556_thamno AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
30686_cyathophylla AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
33413_thamno AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
35236_rex AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
35855_rex AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
38362_rex AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
39618_rex AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
40578_rex AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
41478_cyathophylloides AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
41954_cyathophylloides AGTTGAGCTATAGCTGAATTAGACTGCTGTGAGTTTGATACTGATGAGAACTGTATGGCAT  
// * * [17]  
29154_superba AGCAAATTGCTTGGCCCTGAGAAAATGTAACAGCTTATAAACGCCAGTTCTTATTGGCTC  
30556_thamno AGCAAATTGCTTGGCCCTGAG- AAATGTAACAGCTTATAAACGCCAGTTCTTATTGGCTC  
30686_cyathophylla AGCAAATTGCTTGGCCCTGAG- AAATGTAACAGCTTATAAACGCCAGTTCTTATTGGCTC  
32082_przewalskii AGCAAATTGCTTGGCCCTGAG- AAATGTAACAGCTTATAAACGCCAGTTCTTATTGGCTC  
35236_rex AGCAAATTGCTTGGCCCTGAG- AAATGTAACAGCTTATAAACGCCAGTTCTTATTGGCTC  
35855_rex AGCAAATTGCTTGGCCCTGAG- AAATGTAACAGCTTATAAACGCCAGTTCTTATTGGCTC  
38362_rex AGCAAATTGCTTGGCCCTGAG- AAATGTAACAGCTTATAAACGCCAGTTCTTATTGGCTC  
40578_rex AGCAAATTGCTTGGCCCTGAG- AAATGTAACAGCTTATAAACGCCAGTTCTTATTGGCTC  
41478_cyathophylloides AGCAAATTGCTTGGCCCTGAG- AAATGTAACAGCTTATAAACGCCAGTTCTTATTGGCTC  
41954_cyathophylloides AGCAAATTGCTTGGCCCTGAG- AAATGTAACAGCTTATAAACGCCAGTTCTTATTGGCTC  
// * * [17]
```

RAD-seq genomics

The command-line interface (CLI)

ipyrad analyses can be run using a simple command line tool to run steps of the assembly.

The params file

A single params file describes all the parameters needed to assemble a dataset from steps 1 - 7 .

```
ipyrad -p params-data.txt -s 1234567|
```

RAD-seq genomics

The parameter settings:

<http://ipyrad.readthedocs.io/parameters.html>

We will walk through the most important of these. They include options for entering the location of your raw data, and where you want the assembled data to be stored, as well as filtering options, and parameters that affect the efficiency and accuracy of assemblies.

When working with the command line you can create a default params file with the -n argument and then edit it with a text editor.

```
# create a new params file for 'new'  
ipyrad -n new  
  
# edit it with a text editor  
subl params-new.txt
```

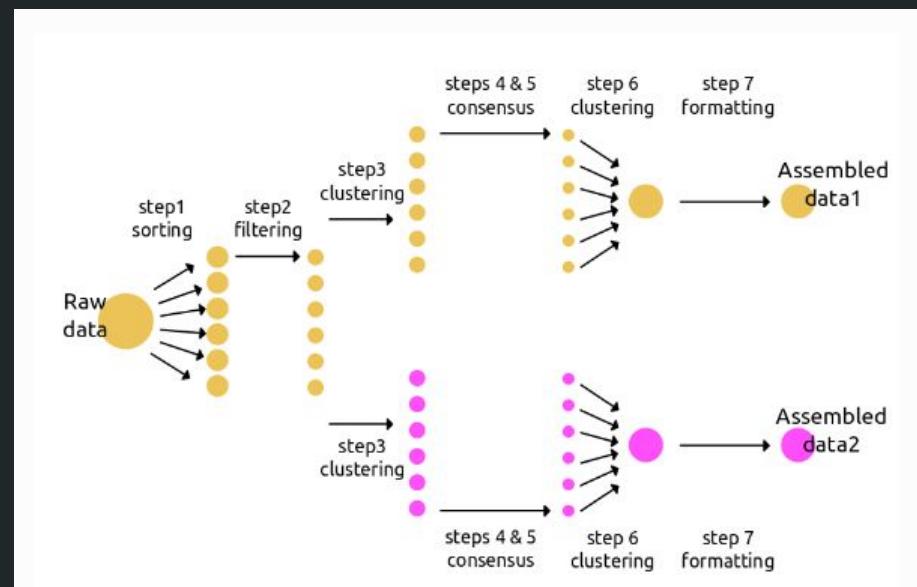
```
|----- ipyrad params file (v.0.7.23)-----  
new      ## [0] [assembly_name]: Assembly name. Used to na  
. /      ## [1] [project_dir]: Project dir (made in curdir)  
## [2] [raw_fastq_path]: Location of raw non-demul  
## [3] [barcodes_path]: Location of barcodes file  
## [4] [sorted_fastq_path]: Location of demultiplexed  
## [5] [assembly_method]: Assembly method (denovo or  
denovo   ## [6] [reference_sequence]: Location of reference  
## [7] [datatype]: Datatype (see docs): rad, gbs,  
rad      ## [8] [restriction_overhang]: Restriction overhang  
TGCAG,  ## [9] [max_low_qual_bases]: Max low quality base  
5       ## [10] [phred_0_score_offset]: phred 0 score offset  
33      ## [11] [mindepth_statistical]: Min depth for statistical  
6       ## [12] [mindepth_majorrule]: Min depth for majority rule  
6       ## [13] [maxdepth]: Max cluster depth within sample  
10000   ## [14] [clust_threshold]: Clustering threshold in  
0.85    ## [15] [max_barcode_mismatch]: Max number of all  
0       ## [16] [filter_adapters]: Filter for adapters/poly  
0       ## [17] [filter_min_trim_len]: Min length of reads after  
35      ## [18] [max_alleles_consens]: Max alleles per consensus  
2       ## [19] [max_Ns_consens]: Max N's (uncalled bases) per  
5, 5    ## [20] [max_Hs_consens]: Max Hs (heterozygotes) per  
8, 8    ## [21] [min_samples_locus]: Min # samples per locus  
4       ## [22] [max_SNPs_locus]: Max # SNPs per locus (FST)  
20, 20  ## [23] [max_Indels_locus]: Max # of indels per locus  
8, 8    ## [24] [max_shared_Hs_locus]: Max # heterozygous Hs per locus  
0.5     ## [25] [trim_reads]: Trim raw read edges (R1>, -R2<)  
0, 0, 0, 0 ## [26] [trim_loci]: Trim locus edges (see docs)  
0, 0, 0, 0 ## [27] [output_formats]: Output formats (see docs)  
p, s, v  ## [28] [pop_assign_file]: Path to population assignment file
```

x - □ deren@oud:~/PDSB/12-parallel-genomics
deren@oud:~/PDSB/12-parallel-genomics (master)\$

Branching

A way to continue an assembly from a checkpoint and apply a different set of parameters.

In the end you will have multiple assemblies that can be compared.



x - □ deren@oud:~/PDSB/12-parallel-genomics
deren@oud:~/PDSB/12-parallel-genomics (master)\$

Ipyrad API

The ipyrad API let's you do all of these same things but in a way that is much cleaner, and more programmatic and reproducible.

http://ipyrad.readthedocs.io/API_user-guide.html

Copy notebook from the Habanero cluster:

```
cp /rigel/edu/w4050/files/nb-12-example.ipynb ~
```

Setting parameters of the assembly

```
data = ip.Assembly("Pedicularis")
```

New Assembly: Pedicularis

```
data.set_params("project_dir", "/rigel/edu/w4050/files/analysis-ipyrad")
data.set_params("sorted_fastq_path", "/rigel/edu/w4050/files/example.emp")
data.set_params("clust_threshold", 0.90)
data.set_params("mindepth_majrule", 10)
data.set_params("mindepth_statistical", 10)
data.set_params("filter_adapters", 2)
data.set_params("output_formats", "*")
```

Run ipyrad assembly

```
data.run("12", ipyclient=ipyclient, show_cluster=True)
```

```
host compute node: [24 cores] on node009
Assembly: Pedicularis
[#####] 100% loading reads | 0:00:04 | s1 |
[#####] 100% processing reads | 0:01:09 | s2 |
```

Ipyrad API

Using ipyrad with jupyter and ipyparallel

http://ipyrad.readthedocs.io/HPC_Tunnel.html

cp /rigel/edu/w4050/files/nb-12-example.ipynb ~

Setting parameters of the assembly

```
data = ip.Assembly("Pedicularis")
```

New Assembly: Pedicularis

```
data.set_params("project_dir", "/rigel/edu/w4050/files/analysis-ipyrad")
data.set_params("sorted_fastq_path", "/rigel/edu/w4050/files/example.emp")
data.set_params("clust_threshold", 0.90)
data.set_params("mindepth_majrule", 10)
data.set_params("mindepth_statistical", 10)
data.set_params("filter_adapters", 2)
data.set_params("output_formats", "*")
```

Run ipyrad assembly

```
data.run("12", ipyclient=ipyclient, show_cluster=True)
```

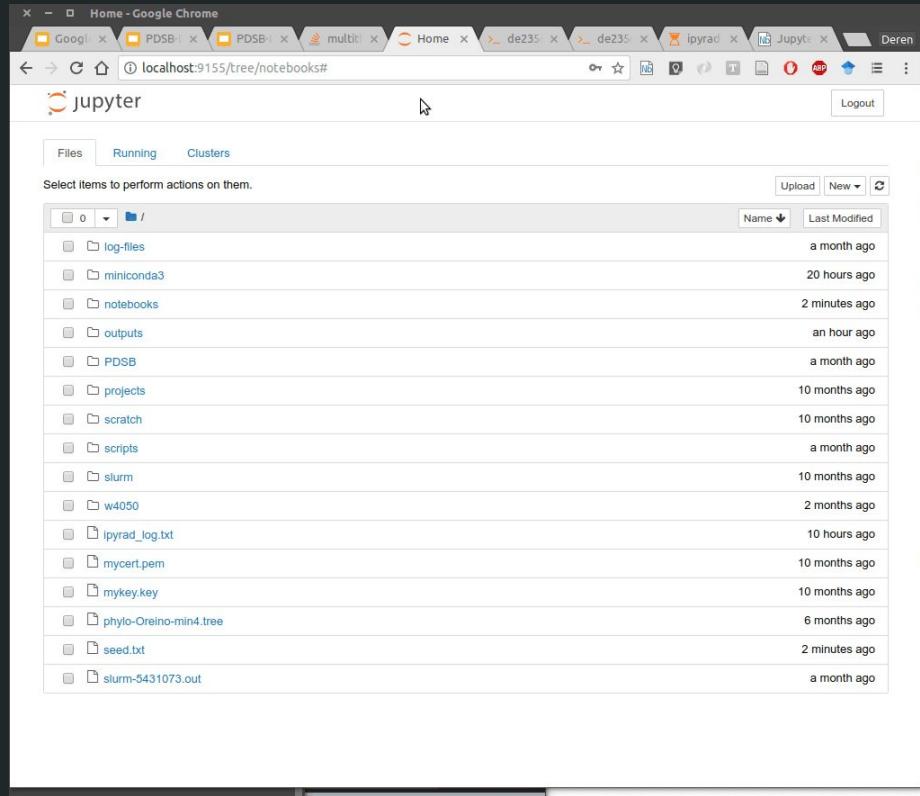
```
host compute node: [24 cores] on node009
Assembly: Pedicularis
[#####] 100% loading reads | 0:00:04 | s1 |
[#####] 100% processing reads | 0:01:09 | s2 |
```

RAD-seq genomics

After submitting jobs to run on ipcluster you can check that your jobs are running correctly by using the `top` command from a terminal in your jupyter dashboard.

Press the ‘q’ key to quit out of top and return to the normal terminal.

The ability to check your running jobs with top makes working interactively on the cluster much easier to use than submitting batch jobs with sbatch.



Assignments notebook is in assignments/
Assigned reading in syllabus:

Assignment: [Link to Session 12 repo](#)

Readings: [See syllabus](#)

Collaborate: Work together in this [gitter chatroom](#)