Programming for Bioinformatics

Hector Corrada Bravo
Center for Bioinformatics and Computational Biology
Fall 2014

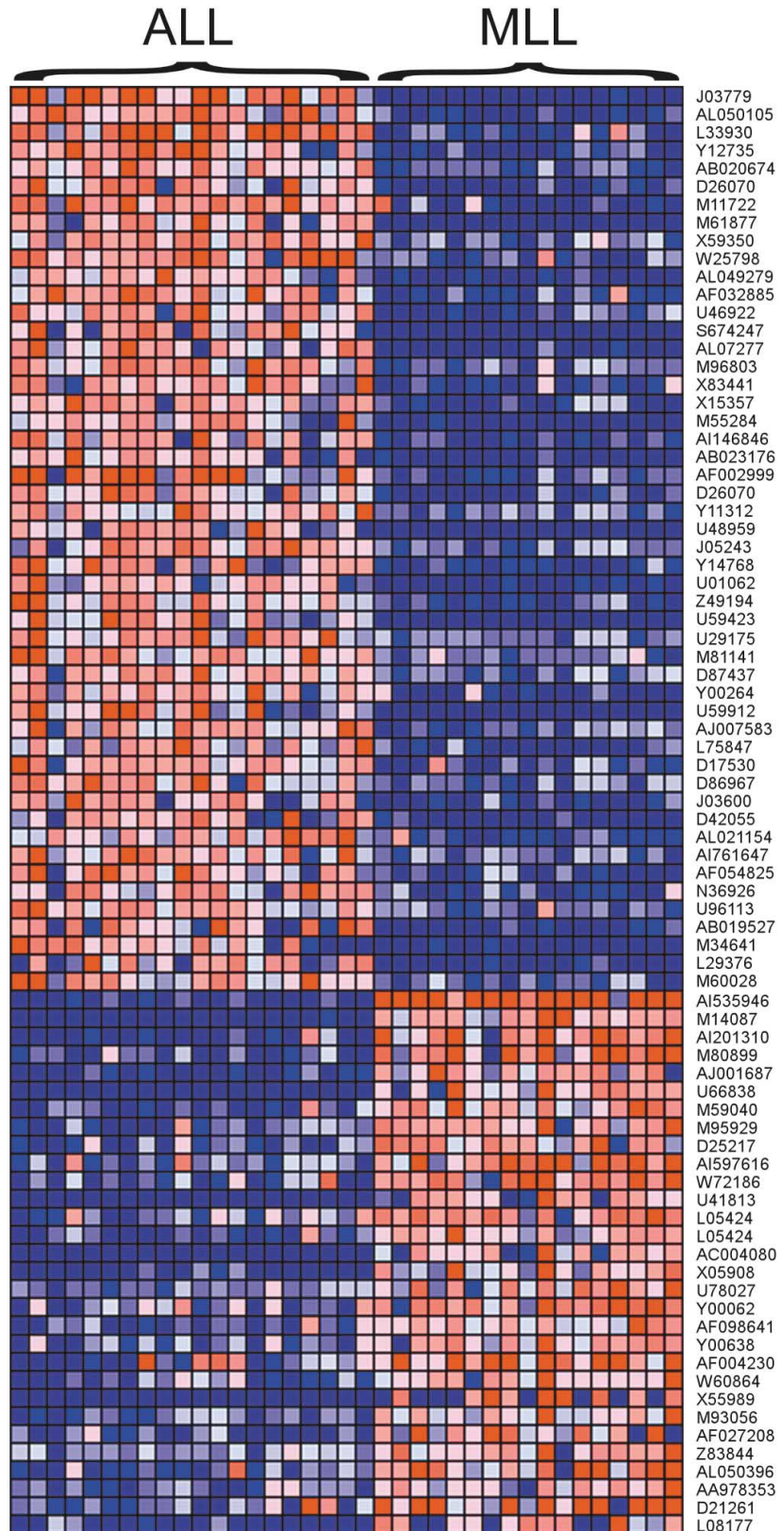
Entities, processes and artifacts

- Genome Sequences

- TCAGTTGGAGCTGCTCCCCCACGGCCTCTCCTCACATTCCACGTCCTGTAGCTCTATGACCTCCACCTTTGAGTCCCTCCTCTCACACCTGACATGAAAAGGCACATGAGGATCCTCAAATACCCC
  GTGATCAGTCTCAGGGTAGCTCTCATAGCCTGGACAGGGCCCCCCTCGGGGGTTGCGCCCAGGTCCAGGCGGGGGATGCACAGCAACAGTCACCGAAGCAGAAGCCGTCACAGTGGTGATGGGCTG
  GCAGTAGCTGGGCACAGAGCTGCCCATGGCGGTGGACGTTGGGTTCCGAGGGTTGTGAGAACGGGCCCCACGGGGCCCTGAGCGGTCCCTATTGCTAGGGCCAGAATGCCCTTCAGTAGAAATTTC
  AAAAGCGTCTCTGCGCGGTCTGTAGGGGGGTGGCCGCAAGCCTTCTCTAGGGGGATCCCTTCGAGGCTGCTGGCCTTGCCGTCCAGGGGACAAGGAGCCAGAGTCCAGGTGGGGCTGTTGCCGAGG
  GGTCAAGGGAGGCTGATGTCTGGAGTCCGGATGGACCACCTGCAGAGGAGAGACATAGGTCAACACAGGGAGGTAGGATGGTGGTGATGTTCCACCCACAAAAGAAAACCTATTCCTTTAGAAACC
  TCCAGGATGTGAATCCTGCCTGCACCTGCACAGCTGGCTGGAGGCATATAGCCACTGCCCATAGATCTCAACTTACCCTCACAACCAACTGCCCCCAGGCCTAAGTTCTCTGCCTCAAAACTGCCA
  AGGCCTGGATAGCCAAGAGCCTGGGTGTCTTGGAAATATGCAACCATAAATAGTAGCTTTTAGAAGTATAAGGCTCCTGTTTCTGGGTCATATTAGTGTTGTTTTCACCTGTCCCCAGCCCTAAGC
  CAGGTGTGGCCAGAAGCAAATGTACTGTAAGAGCAGAGCAAAAACTTCCACACAGATAGTTCTGTTAGGCAATACATCTCTGCCTGACTATTAGGAATCTGGTTTCTGGGTCCTCTGTACAAAGCT
  CGGAGCAACACAGTGGCCACATCAATCAAAAGGACCGTGACCAACTTCAAAGTCGGTGAGCTTGTACCTATTTTTAGGCTCCTGCTGAACAGAACCAGATTCACACTACAGCTCAGCAGGGCATCG
  TCACGGGTGTGTGTGTGTGTGTGTGTGTGTGTGTGTGTGTGTTGGGGGGGGGGGGGTGGACAGAGGACGGGGACACAATTCACTGGCCAGCCCTTCTCTCCTTCAAGGAAGGCTGCTCTAGCCTG
  GGACTGGAATACACATTTCCTGTAAACATGGTGGGGGCCTCAGGCAAGCCAGAGTTTTGGAGCCTTCCTTAACTCTTCAAGGTGAGCATCTTGACTTGGAGGGTGGGGGTGCGGGTAAGGAAGGAA
  CCTGTGGACTCCTCCCTACAAGACAGAAAAGGAATAAGCCACGAAGACAATAACGATTTTTGTATCAAGCGTCCTCTCCCATTTCAGCTTACCTGACAATGAAATCAAATTCGGACCCTGCAAGCA
  TCAGTACACCCAGCAGAGTGGACACAGCACCGTCCAGAACGGGAGCAAACATGTGCTCCAGAGCGAGCATAGCCCTGTGGTTCTTGTCCCCAATGGCTGTCAGAAAGGCCTGAACAAAGGAGAAAA
  TTGACACGGTCACATTCTGGGTGTGGTAAAGTGCTCAGCTGTGTCTATACTTGGGTTTTGTAT…

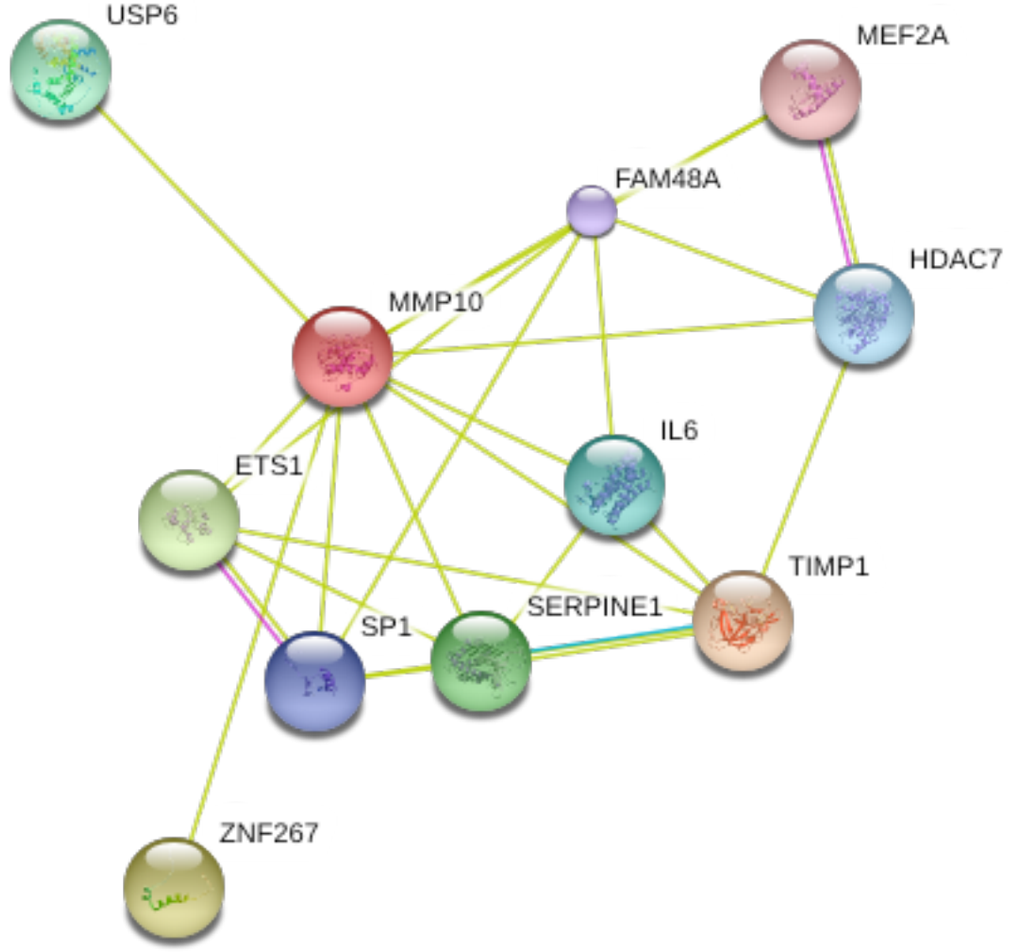
Entities, processes and artifacts

- Genome Sequences
- Gene Expression Measurements



Entities, processes and artifacts

- Genome Sequences
- Gene Expression Measurements
- Networks of gene or protein relationships/interactions



X

Entities, processes and artifacts

- Genome Sequences
- Gene Expression Measurements
- Networks of gene or protein relationships/interactions
- Sequence alignments
- Phylogenetic trees
- Genome variation in populations

Libraries

1. Connect/access databases

2. Data structures for fundamental objects

3. Basic operations/algorithms on these structures

4. Tools for communication

Libraries

- R: Bioconductor: http://bioconductor.org/
- Python: BioPython: http://biopython.org/wiki/Main_Page
- C++: SeqAn: http://www.seqan.de/
- Perl: BioPerl: http://www.bioperl.org/wiki/Main_Page
- Ruby: BioRuby: http://www.bioruby.org/
- Java: BioJava: http://biojava.org/wiki/Main_Page

Databases

We will discuss these in later lectures

- Sequence: Genbank/Refseq/Unigene/Short Read Archive
- Gene Expression: Gene Expression Omnibus
- Pathways: KEGG
- Function: Gene Ontology

Standards

- Many of these data are stored in standard formats:
    - FASTA sequence format
    - FASTQ sequence/with quality
    - GTF/GFF for genomic features (genes, exons, introns, etc.)
- Libraries provide interfaces to the databases and standards.

Encapsulation

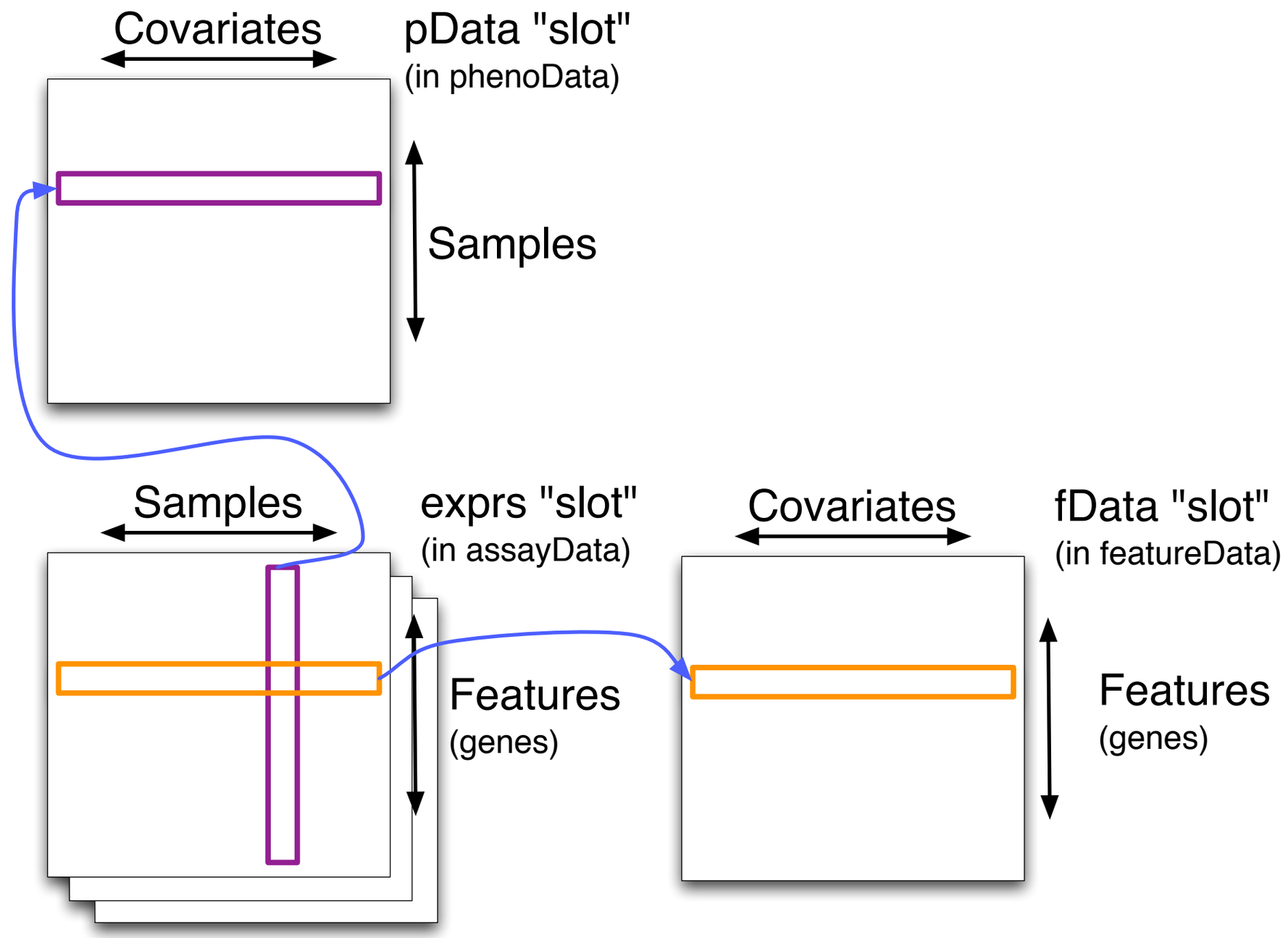
- Libraries also encapsulate these standard data types into appropriate data structures for the given language.

- Example: sequence records in BioPython

- Example: ‘GenomicRanges’ in R/Bioconductor

Encapsulation

- Example: ‘ExpressionSet’ in R/Bioconductor

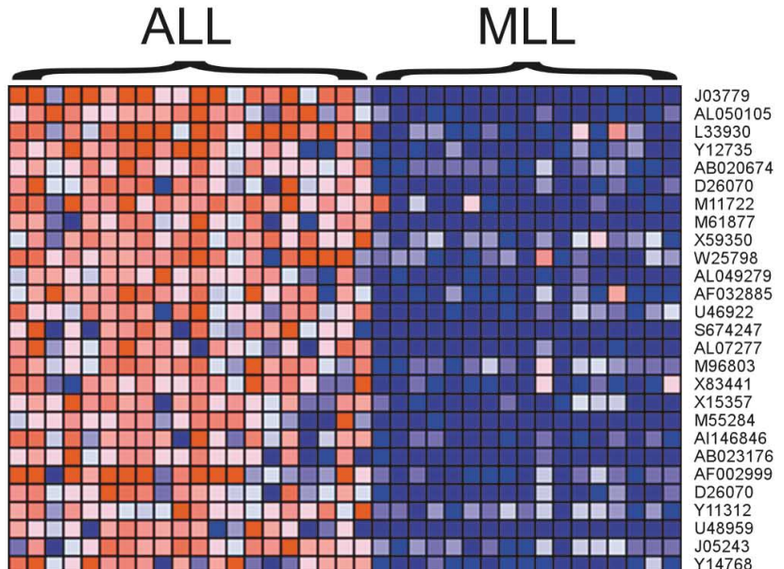


Encapsulation

- Basic operations on these data structures
  - Standard computation: e.g., aggregation, filtering, etc.
  - Bio-specific: e.g., genomic region overlap, DNA->AminoAcid translation

Communication

- Big part of the Bioinformatician and Computational Biologist job:
  - **Communicate results**
- Examples:
  - New sequence aligner: how fast is it? how well does it align?
  - Expression analysis: Does the data match your analvsis?



Communication

- Visualize data
  - Tons of plotting utilities in R/Bioconductor
  - matplotlib in python
- Documentation standards
  - pydoc

Reproducibility

- Extremely important aspect of data analysis
  - ‘Starting from the same raw data, can we reproduce your analysis and obtain the same results?’
- Using libraries helps:
  - Since you don’t reimplement everything, reduce programmer error
  - Large user bases serve as ‘watchdog’ for quality and correctness
- Standard practices help:
  - Version control: git
  - Unit testing: pyunit, RUnit
  - Share and publish: github

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition
  - Algorithm/tool development
  - Computational analysis
  - Communication of results

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up
  - Algorithm/tool development
  - Computational analysis
  - Communication of results

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up
  - Algorithm/tool development: if new analysis tools are required
  - Computational analysis
  - Communication of results

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up
  - Algorithm/tool development: if new analysis tools are required
  - Computational analysis: use tools to analyze data
  - Communication of results

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up
  - Algorithm/tool development: if new analysis tools are required
  - Computational analysis: use tools to analyze data
  - Communication of results: prepare summaries of experimental results, plots, publication, upload processed data to repositories

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up
  - Algorithm/tool development: if new analysis tools are required
  - Computational analysis: use tools to analyze data
  - Communication of results: prepare summaries of experimental results, plots, publication, upload processed data to repositories

Rarely does a single language handle all of these equally well

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up
  - Algorithm/tool development: if new analysis tools are required
  - Computational analysis: use tools to analyze data
  - Communication of results: prepare summaries of experimental results, plots, publication, upload processed data to repositories

Choose the best tool
for the job!

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up

R, python or shell scripting

  - Algorithm/tool development: if new analysis tools are required
  - Computational analysis: use tools to analyze data
  - Communication of results: prepare summaries of experimental results, plots, publication, upload processed data to repositories

Practical Tips

- Many tasks can be organized in modular manner:

  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up

  - Algorithm/tool development: if new analysis tools are required

C/C++, R or python (depending on task)

  - Computational analysis: use tools to analyze data

  - Communication of results: prepare summaries of experimental results, plots, publication, upload processed data to repositories

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up
  - Algorithm/tool development: if new analysis tools are required
  - Computational analysis: use tools to analyze data

Best managed as shell or python scripts

  - Communication of results: prepare summaries of experimental results, plots, publication, upload processed data to repositories

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up
  - Algorithm/tool development: if new analysis tools are required
  - Computational analysis: use tools to analyze data
  - Communication of results: prepare summaries of experimental results, plots, publication, upload processed data to repositories

I use R almost exclusively

Practical Tips

- Many tasks can be organized in modular manner:
  - Data acquisition: get data, put it in usable format (many ‘join’ operations), clean it up
  - Algorithm/tool development: if new analysis tools are required
  - Computational analysis: use tools to analyze data
  - Communication of results: prepare summaries of experimental results, plots, publication, upload processed data to repositories

Usually all of this is managed by a *pipeline* of shell/python scripts

Practical Tips

- Modularity requires organization and careful thought
- In bioinformatics we wear two hats
  - Algorithm/tool developer
  - **Experimentalist**: we don’t get trained to think this way enough!
- It helps two consciously separate these two jobs

Think like an experimentalist

- Plan your experiment
- Gather your raw data
- Gather your tools
- Execute experiment
- Analyze
- Communicate

Think like an experimentalist

- Let this guide your organization. I find structuring my projects like this to be useful:

```
project/
| data/
| | processing_scripts
| | raw/
| | proc/
| tools/
| | src/
| | bin/
| exps
| | pipeline_scripts
| | results/
| | analysis_scripts
| | figures/
```

Think like an experimentalist

- Keep a lab notebook!

- Literate programming tools are making this easier for computational projects

  - http://en.wikipedia.org/wiki/Literate_programming

  - http://ipython.org/notebook.html

  - http://www.rstudio.com/ide/docs/r_markdown

Think like an experimentalist

- Separate experiment from analysis from communication
    - Store results of computations, write separate scripts to analyze results and make plots/ tables
- **Aim for reproducibility**
    - There are serious consequences for not being careful
        - Publication retraction
        - Worse: http://videolectures.net/cancerbioinformatics2010_baggerly_irrh/
    - Lots of tools available to help, use them! Be proactive: learn about them on your own!