

BEDTools– Genome Arithmetic

Biol4230 Thurs, April 13, 2017
Bill Pearson wrp@virginia.edu 4-2818 Jordan 6-057

- Borrowed from Aaron Quinlan, BEDTools author
- Genome file types/formats
 - format types
 - BED files
- Overview of genome features
- Genome arithmetic
 - approaches
 - UCSC "binning" algorithm
- Introduction to BEDtools
 - intersection / window / many more

fasta.bioch.virginia.edu/biol4230

1

To learn more:

File formats:

1. <http://genome.ucsc.edu/FAQ/FAQformat>
2. Mills (2014) *Curr. Protoc. Bioinform.* 45:A.1B.1-A.1B.18

BEDTools:

1. <http://bedtools.readthedocs.org>
2. Quinlan, A. R. BEDTools: The Swiss-Army Tool for Genome Feature Analysis. *Curr Protoc Bioinformatics* 47, 11.12.1–11.12.34 (2014).
3. Galaxy Screencasts: <http://main.g2.bx.psu.edu/>
4. Download genome features from UCSC and use BEDTools on the command line

fasta.bioch.virginia.edu/biol4230

2

Analysis of transcription regulation

1. Affy-chip/RNA-seq for RNA expression levels
2. Identify differentially expressed genes (edgeR, DESeq2)
 - (possibly) identify subsets of genes associated with different pathways
3. Isolate regions of DNA around (pathway-specific) coordinately expressed (induced) genes
 - BEDtools
4. Take collections of candidate regulatory regions and look for common DNA motif
 - meme, HOMER

fasta.bioch.virginia.edu/biol4230

3

Genome file formats:

| Data | Alignment | Feature |
|---------|-----------|------------|
| GenBank | SAM/BAM | GFF2/GTF |
| EMBL | Axt | GFF3 |
| FASTA | MAF | VCF |
| FASTQ | Chain | BED/bigBED |
| | Stockholm | bedGraph |
| | | WIG/bigWIG |

Mills (2014) Current Prot. in Bioinfo.
Appendix A.1B.1-A.1B.18

fasta.bioch.virginia.edu/biol4230

4

Genome data file formats:

- **FASTA:**

```
>sp|P09488|GSTM1_HUMAN  
MPMILGYWDIRGLAHAIRLLEYTDSSYYEKKYTMGDAPDYDRSQWLNEKFKLGLDFPNLPYLI  
DGAHKITQSNAILCYIARKHNLCGETEEEKIRVDILENQTMNDNMQLGMICYNPEF
```

- **FASTQ (FASTA with quality scores)**

```
@UNC10-SN254_238:4:1101:1351:51174/1  
CTTCGCGGTAGCTGGACCGCCGTTAGTCGCNAATANGCNGCTCTNTGT  
+  
B@CFFFFFFDFHHHJJJJJJJJJ@PHHJG1JJ#####  
@UNC10-SN254_238:4:1101:1351:7840/1  
ATTTTTTACATGGAGCAGGAACTGGAGTAAATGCAANACNGTGTNTAA  
+  
CCCCFFFFHHHHHJIIJJIIGIIJHIGGHIIIHIG#####
```

Mills (2014) Current Prot. in Bioinfo.
Appendix A.1B.1-A.1B.18

fasta.bioch.virginia.edu/biol4230

5

Genome feature file formats:

- **GFF/GTF/GFF2:**

```
##gff-version 2  
##type Protein  
  
##sequence-region P09488 1 218  
P09488 UniProtKB mature_protein_region 2 218 . . . Note "Glutathione S-trans. Mu 1"  
P09488 UniProtKB polypeptide_domain 2 88 . . . Note "GST N-terminal"  
P09488 UniProtKB polypeptide_domain 90 208 . . . Note "GST C-terminal"  
P09488 UniProtKB polypeptide_region 7 8 . . . Note "Glutathione binding"  
P09488 UniProtKB polypeptide_region 46 50 . . . Note "Glutathione binding"  
P09488 UniProtKB polypeptide_region 59 60 . . . Note "Glutathione binding"  
P09488 UniProtKB polypeptide_region 72 73 . . . Note "Glutathione binding"  
P09488 UniProtKB binding_motif 116 116 . . . Note "Substrate"  
P09488 UniProtKB alt_seq_site 153 189 . . . Note "UniProtKB FT ID: VSP_036618"  
P09488 UniProtKB nat_variant_site 173 173 . . . Note "K -> N"  
P09488 UniProtKB nat_variant_site 210 210 . . . Note "S -> T" ; ...  
  
Note field[8] is can be very long, separated by ';' ;  
Note "S -> T" ; Note "UniProtKB FT ID: VAR_014497" ; Note "dbSNP:rs449856" ; Link  
"http://www.ncbi.nlm.nih.gov/SNP/snp_ref.cgi?type=rs&rs=449856" ; Link  
"http://www.ensembl.org/Homo_sapiens/Variation/Explore?v=rs449856"
```

Mills (2014) Current Prot. in Bioinfo.
Appendix A.1B.1-A.1B.18

fasta.bioch.virginia.edu/biol4230

6

Genome feature file formats: (BED)

| chr | start 0-based | stop 1-based | name (len = stop - start) | score | strand |
|-----|------------------|-----------------|------------------------------|-------|--------|
|-----|------------------|-----------------|------------------------------|-------|--------|

BED files:

```

chr1    110230417 110230531 NM_146421_exon_0_0_chr1_110230418_f    0      +
chr1    110230791 110230867 NM_146421_exon_1_0_chr1_110230792_f    0      +
chr1    110231294 110231359 NM_146421_exon_2_0_chr1_110231295_f    0      +
chr1    110231669 110231751 NM_146421_exon_3_0_chr1_110231670_f    0      +
chr1    110231846 110231947 NM_146421_exon_4_0_chr1_110231847_f    0      +
chr1    110232892 110232988 NM_146421_exon_5_0_chr1_110232893_f    0      +
chr1    110235827 110236367 NM_146421_exon_6_0_chr1_110235828_f    0      +
chr1    110230417 110230531 NM_000561_exon_0_0_chr1_110230418_f    0      +
chr1    110230791 110230867 NM_000561_exon_1_0_chr1_110230792_f    0      +
chr1    110231294 110231359 NM_000561_exon_2_0_chr1_110231295_f    0      +
chr1    110231669 110231751 NM_000561_exon_3_0_chr1_110231670_f    0      +
chr1    110231846 110231947 NM_000561_exon_4_0_chr1_110231847_f    0      +
chr1    110232892 110232988 NM_000561_exon_5_0_chr1_110232893_f    0      +
chr1    110233075 110233186 NM_000561_exon_6_0_chr1_110233076_f    0      +
chr1    110235827 110236367 NM_000561_exon_7_0_chr1_110235828_f    0      +

```

Mills (2014) Current Prot. in Bioinfo.
Appendix A.1B.1-A.1B.18

fasta.bioch.virginia.edu/biol4230

7

BED (Browser Extensible Data) format:

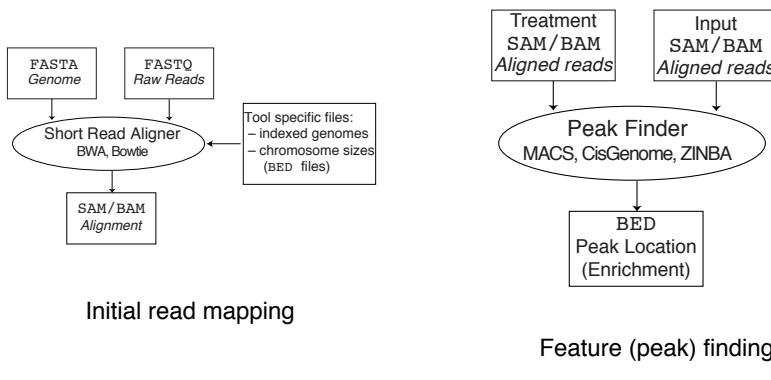
| Column | Field | Description |
|--------|-------------|---|
| 1 | Chromosome | Chr1, chr2... chrX or scaffold |
| 2 | Start | Zero based start of feature |
| 3 | Stop | One based end of a feature, not inclusive |
| 4 | Name | Name displayed in the genome browser |
| 5 | Score | 0-100 |
| 6 | Strand | “+” or “-” |
| 7 | thickStrand | Beginning of thick band in browser |
| 8 | thickEnd | End of thick band in browser |
| 9 | itemRGB | Colors the feature using RGB codes |
| 10 | blockCount | Number of block in the BED line |
| 11 | blockSizes | Comma separated list of block sizes |
| 12 | blockStarts | Comma separated list of block positions relative to the Start |

Mills (2014) Current Prot. in Bioinfo.
Appendix A.1B.1-A.1B.18

fasta.bioch.virginia.edu/biol4230

8

Genome analysis pipelines



Mills (2014) Current Prot. in Bioinfo.
Appendix A.1B.1-A.1B.18

fasta.bioch.virginia.edu/biol4230

9

Genome data (UCSC table browser)

Table Browser

Use this program to retrieve the data associated with a track in text format, to calculate intersections between tracks, and to retrieve DNA sequence covered by a track. For help in using this application see [Using the Table Browser](#) for a description of the controls in this form, the [User's Guide](#) for general information and sample queries, and the [OpenHelix Table Browser tutorial](#) for a narrated presentation of the software features and usage. For more complex queries, you may want to use [Galaxy](#) or our [public MySQL server](#). To examine the biological function of your set through annotation enrichments, send the data to [GREAT](#). Send data to [GenomeSpace](#) for use with diverse computational tools. Refer to the [Credits](#) page for the list of contributors and usage restrictions associated with these data. All tables can be downloaded in their entirety from the [Sequence and Annotation Downloads](#) page.

clade: Mammal genome: Human assembly: Dec. 2013 (GRCh38/hg38)

group: Genes and Gene Predictions track: UCSC Genes add custom tracks track hubs

table: knownGene describe table schema

region: genome position chr1:109631271-109750270 lookup define regions

identifiers (names/acccessions): paste list upload list

filter: create

intersection: create

correlation: create

output format: BED - browser extensible data Send output to Galaxy GREAT GenomeSpace

output file: (leave blank to keep output in browser)

file type returned: plain text gzip compressed

get output summary/statistics

To reset all user cart settings (including custom tracks), [click here](#).

fasta.bioch.virginia.edu/biol4230

10

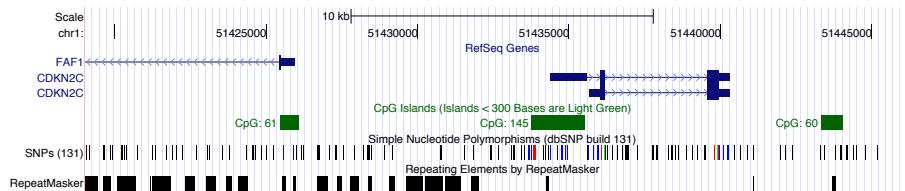
What is a genome “feature”?

- Genes: exons, introns, UTRs, promoters
- Conservation
- Genetic variation
- Transposons
- Origins of replication
- TF binding sites
- CpG islands
- Segmental duplications
- Sequence alignments
- Chromatin annotations
- Gene expression data
- ...
- **Your own observations: put them in context**

Genome “features”

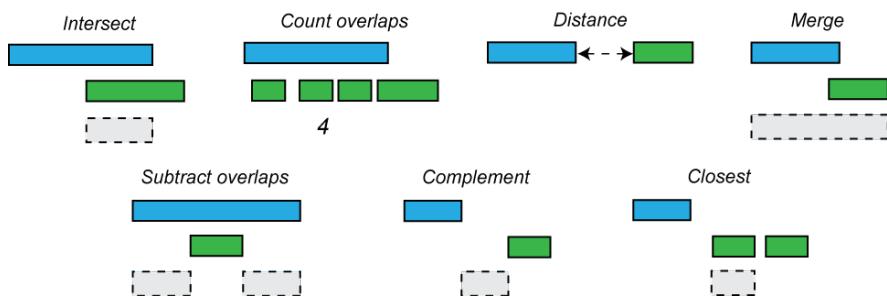


Overlaps, closest, merge...



What is genome arithmetic?

"Set theory on the genome"



Answerable questions

- Closest gene to a ChIP-seq peak.
- Is my latest discovery novel?
- Is there strand bias in my data?
- How many genes does this mutation affect?
- Where did I fail to collect sequence coverage?
- Is my favorite feature significantly correlated with some other feature?

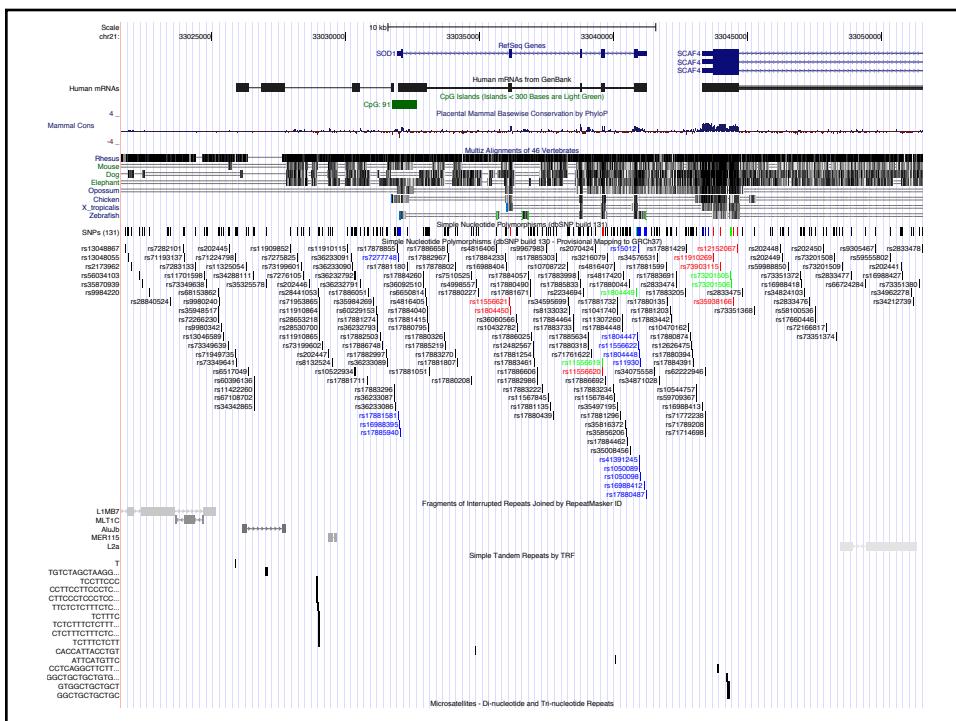
An annoying wealth of formats

- BED
- BEDGRAPH
- WIG
- GFF
- VCF
- SAM/BAM
- ...

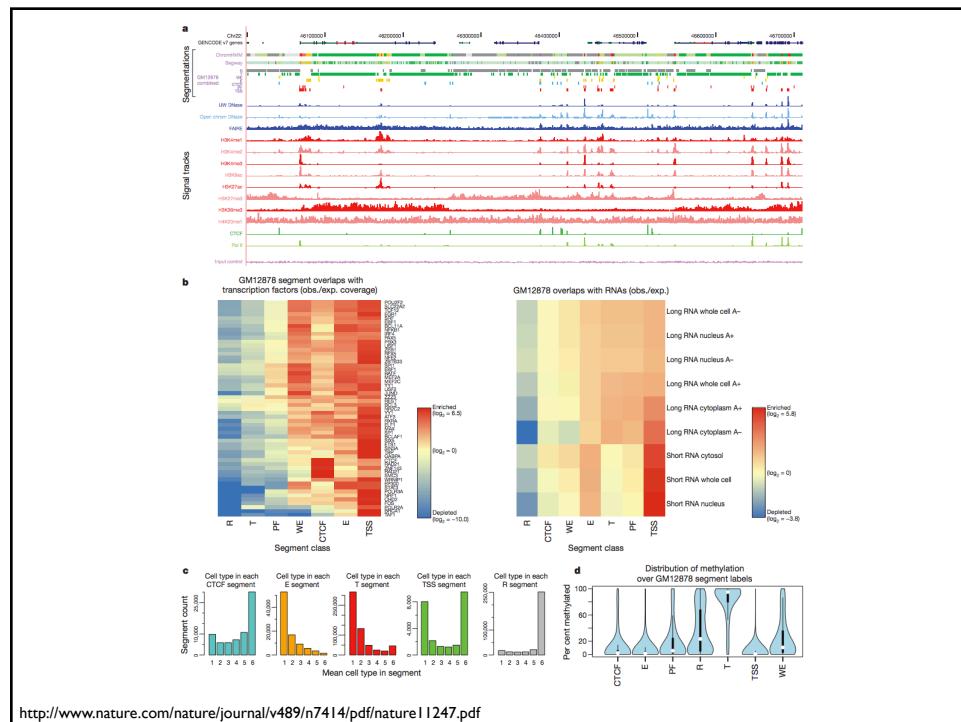
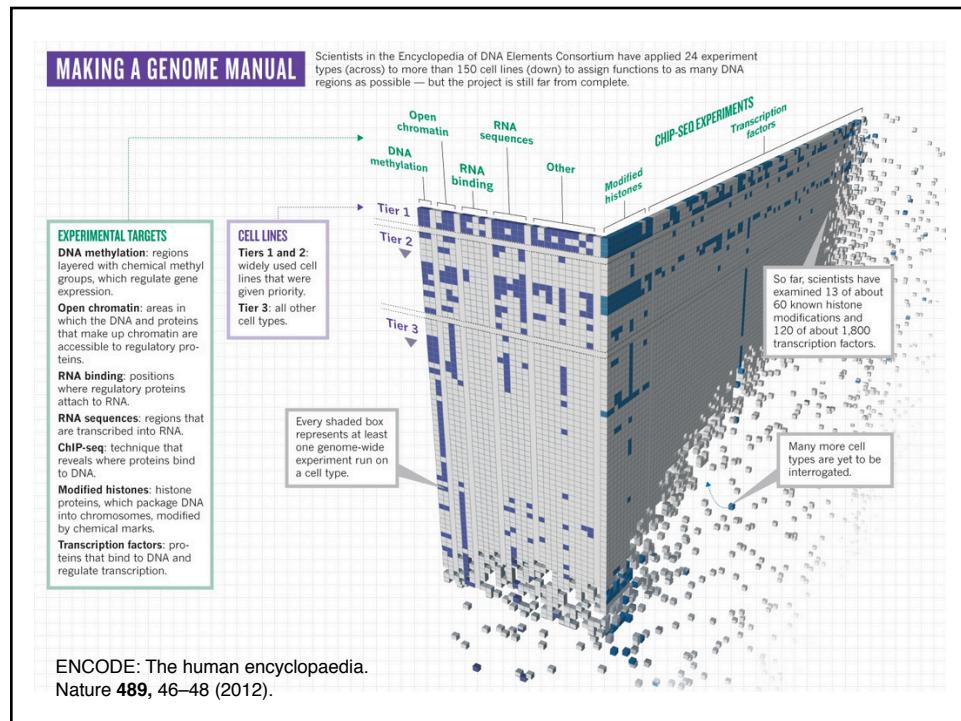
Standard attributes

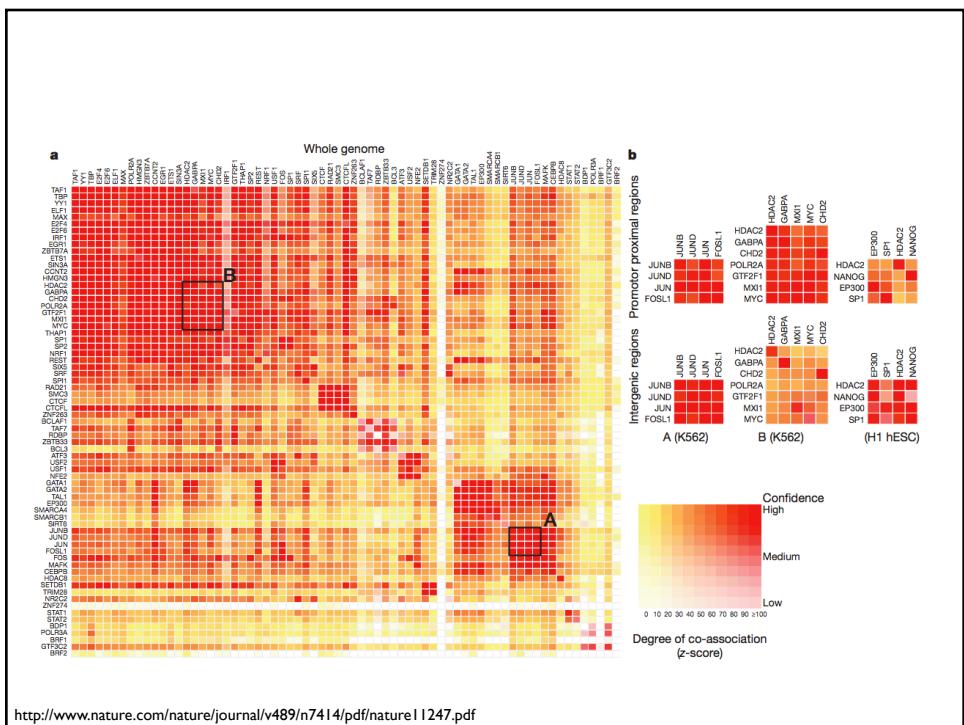
- chrom
- start position
- end position
- name / label
- score / value
- strand
- other

No standards: Some formats use 1-based coordinates, while others use 0-based



Why?





How?

Searching for overlaps

- Brute force: loop over all N features
 - Can be tragically slow when N is large
- Build a “tree”
 - R-tree
 - NCList
 - UCSC “binning” algorithm
- Extend the sort & “merge” algorithm
 - Widely used to “join” database table

Brute force: sloooooow

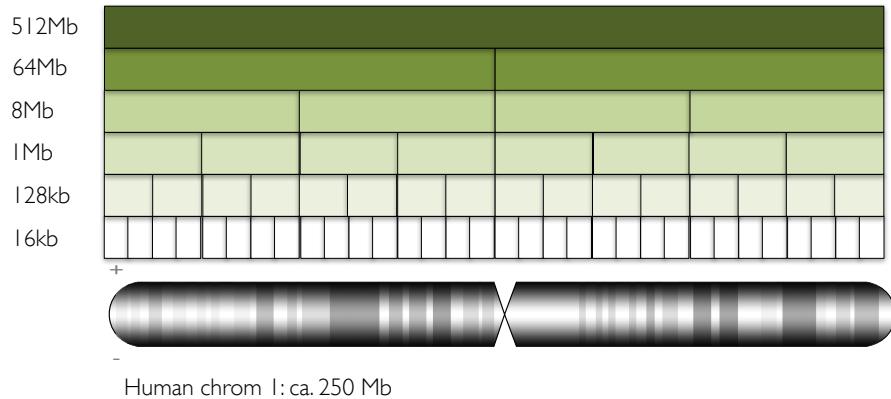
```
foreach fA in file A
{
    foreach fB in file B
    {
        does fA overlap fB?*
    }
}
```

The screenshot shows a terminal window with the title 'overlap.pl'. The script code is as follows:

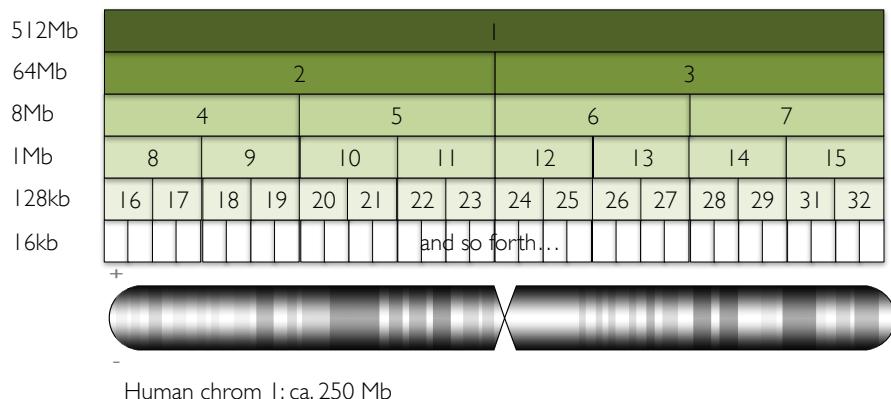
```
1 #!/usr/bin/perl -w
2 use strict;
3
4 my $fileA = shift;
5 my $fileB = shift;
6 my $fileD = shift;
7
8 # attempt to open the files, will if trouble
9 open A, "<$fileA" or die $!;
10 open B, "<$fileB" or die $!;
11
12 my @a_features = split /\n/, read_file($fileA);
13 my @b_features = split /\n/, read_file($fileB);
14
15 foreach my $a (@a_features)
16 {
17     foreach my $b (@b_features)
18     {
19         my ($a_chrom, $a_start, $a_end) = split(":", $a);
20         foreach my $b (@b_features)
21         {
22             my ($b_chrom, $b_start, $b_end) = split(":", $b);
23             next if $a_chrom ne $b_chrom;
24             if ($a_start <= $b_start & $a_end >= $b_start) {
25                 chomp $a;
26                 chomp $b;
27                 print "$a\t\t$b\n";
28             }
29         }
30     }
31 }
32
33 sub overlap {
34     my ($s1, $e1, $s2, $e2) = @_;
35     return min($s1, $e2) - max($s1, $e2);
36 }
37
38 sub max {
39     my ($s1, $s2) = @_;
40     if ($s1 >= $s2) {
41         return $s1;
42     } else {
43         return $s2;
44     }
45 }
46
47 sub min {
48     my ($s1, $s2) = @_;
49     if ($s1 <= $s2) {
50         return $s1;
51     } else {
52         return $s2;
53     }
54 }
```

* When A and B contain many features, the answer will be NO the vast majority of the time.
This means most of the processing is spent NOT finding features that overlap.

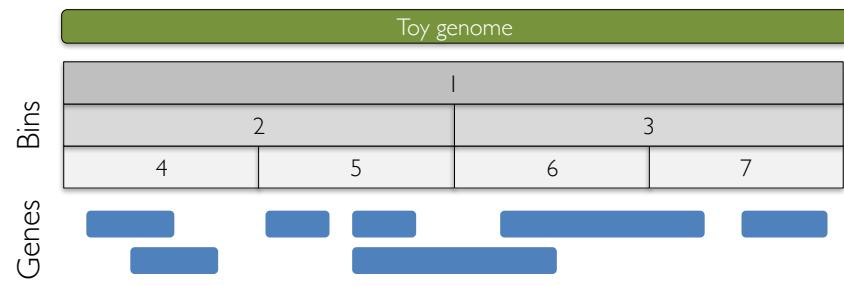
Narrow the search space w/ “binning” *the UCSC genome browser approach*



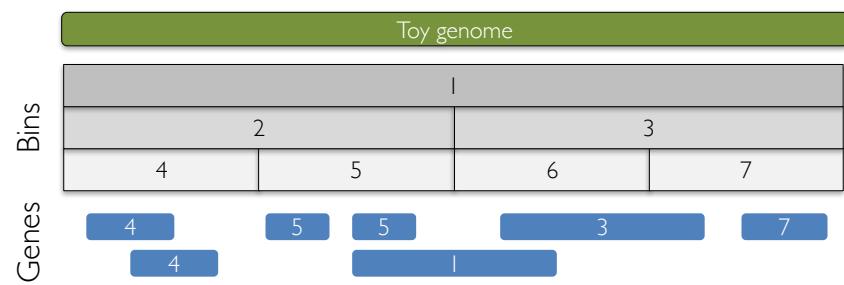
Narrow the search space w/ “binning” *the UCSC genome browser approach*



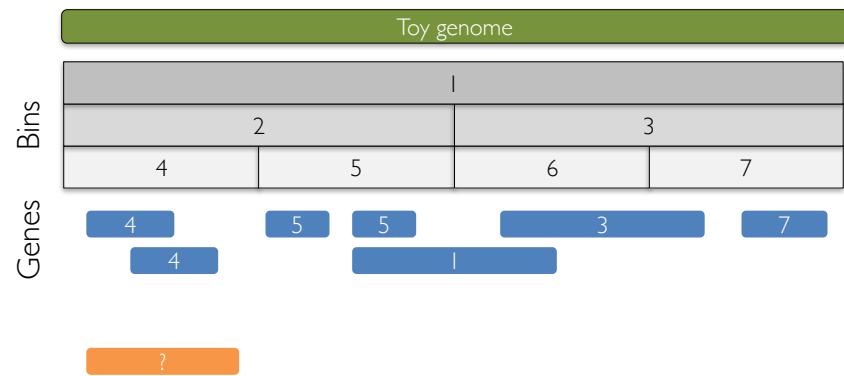
A toy example



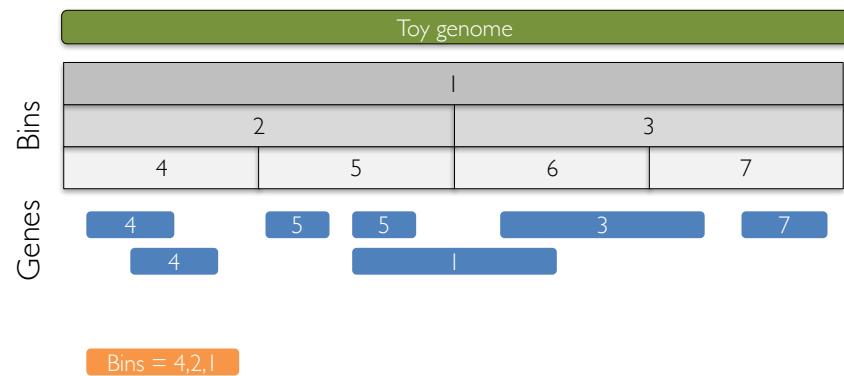
A toy example



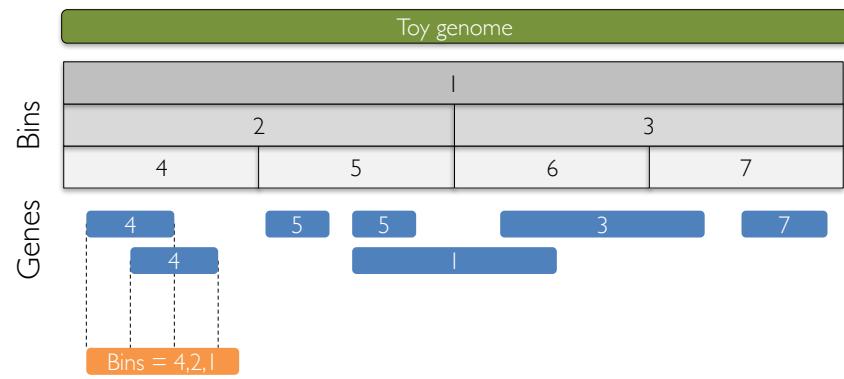
A toy example



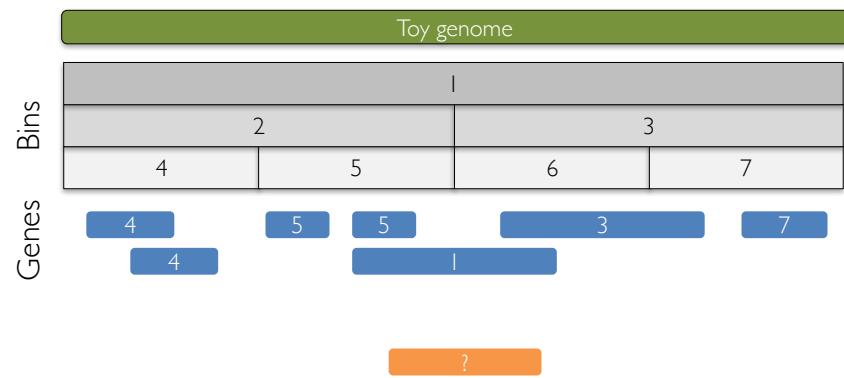
A toy example



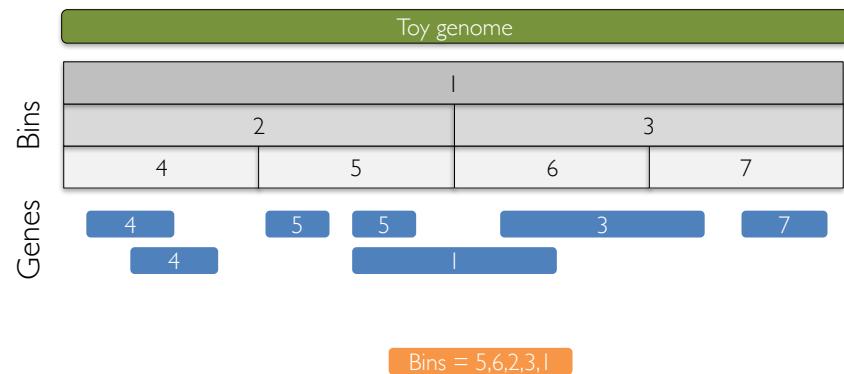
A toy example



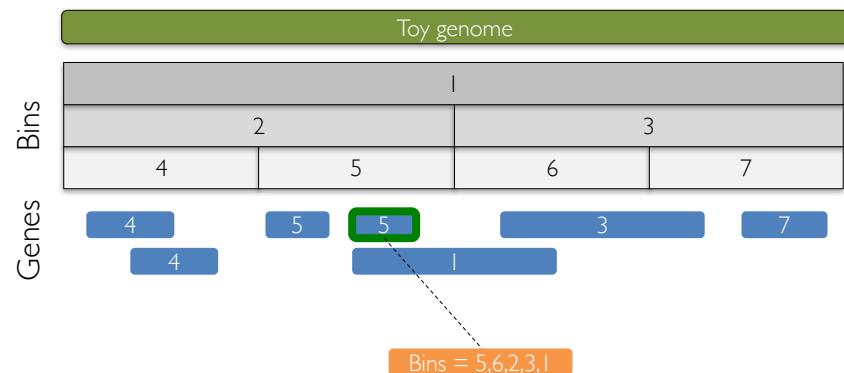
A toy example



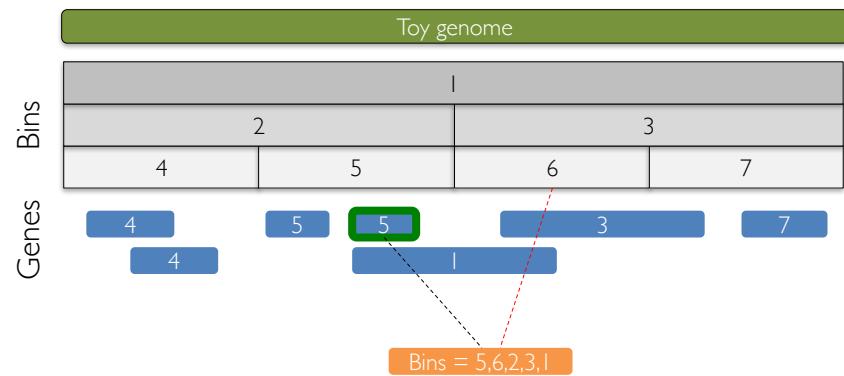
A toy example



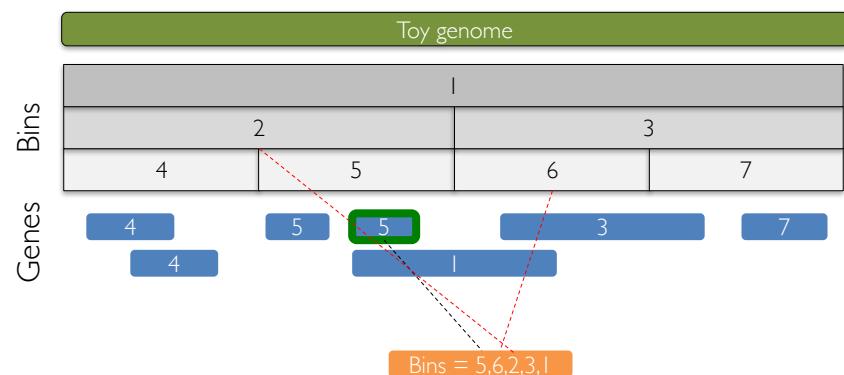
A toy example



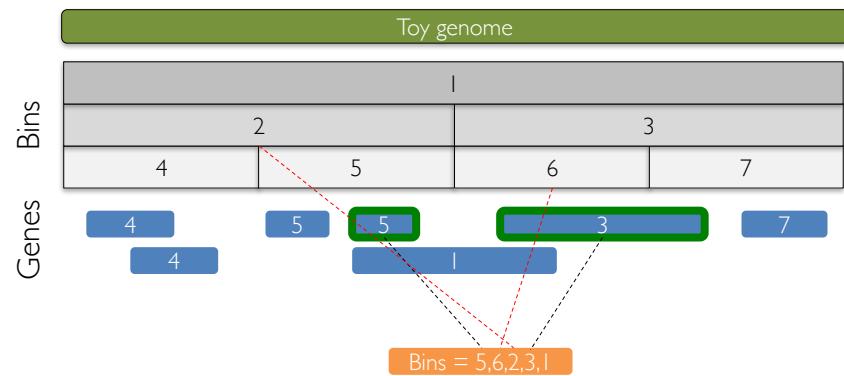
A toy example



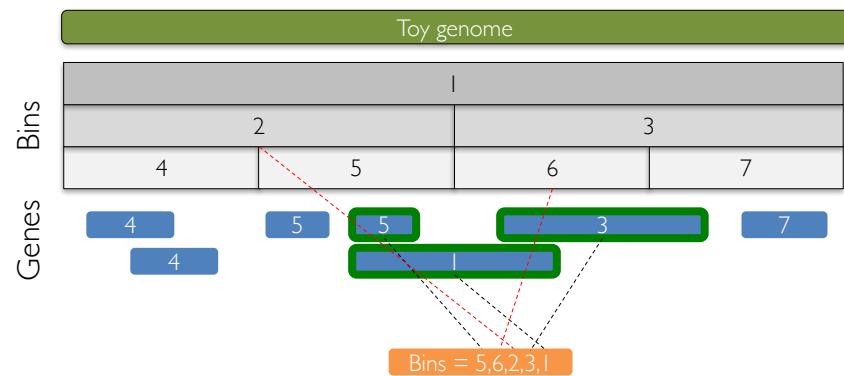
A toy example



A toy example



A toy example

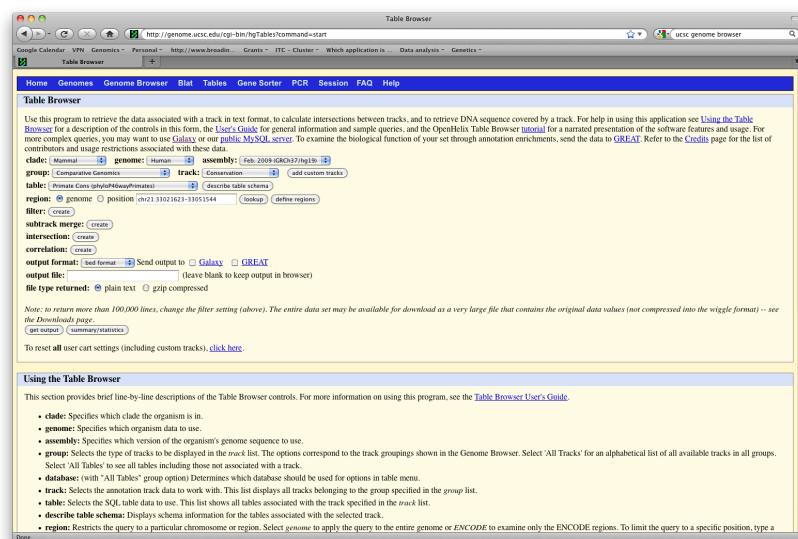


Available tools for GA

[UCSC Genome Browser](#)
[Galaxy](#)
[BEDTools](#)

UCSC Genome Browser

<http://genome.ucsc.edu/cgi-bin/hgTables?command=start>



The screenshot shows the UCSC Table Browser interface. At the top, there's a navigation bar with links like Home, Genomes, Genome Browser, Blat, Tables, Gene Sorter, PCR, Session, FAQ, and Help. Below the navigation bar, the main form has the following fields:

- clade:** Mammal
- genome:** Human
- assembly:** Feb 2009 GRCh37/hg19
- group:** Comparative Genomics
- track:** Conservation
- table:** Primate Cons (phylogenetic)
- region:** genome (checkbox) position chr21:31021623-31051544 (checkbox) (lookup) (define regions)
- filter:** create
- substrate merge:** Create
- intersection:** Create
- correlation:** Create
- output format:** bed format (radio button selected)
- Send output to:** Galaxy (radio button) GREAT (radio button)
- output file:** (leave blank to keep output in browser)
- file type returned:** plain text (radio button selected) gzip compressed (radio button)

Below the form, there's a note about returning more than 100,000 lines and a link to the Downloads page. A "Done" button is at the bottom.

Using the Table Browser

This section provides brief line-by-line descriptions of the Table Browser controls. For more information on using this program, see the [Table Browser User's Guide](#).

- clade:** Specifies which clade the organism is in.
- genome:** Specifies which organism data to use.
- assembly:** Specifies which version of the organism's genome sequence to use.
- group:** Selects the type of tracks to be displayed in the track list. These groups correspond to the track groupings shown in the Genome Browser. Select 'All Tracks' for an alphabetical list of all available tracks in all groups.
- table:** Selects the SQL table or tables to use. This list shows all tables associated with the track specified in the track list.
- describe table schema:** Displays schema information for the tables associated with the selected track.
- region:** Restricts the query to a particular chromosome or region. Select genome to apply the query to the entire genome or ENCODE to examine only the ENCODE regions. To limit the query to a specific position, type a

BEDTools

<https://github.com/arq5x/bedtools2>

bedtools - a swiss army knife for genome arithmetic

Current version: 2.23.0

Note

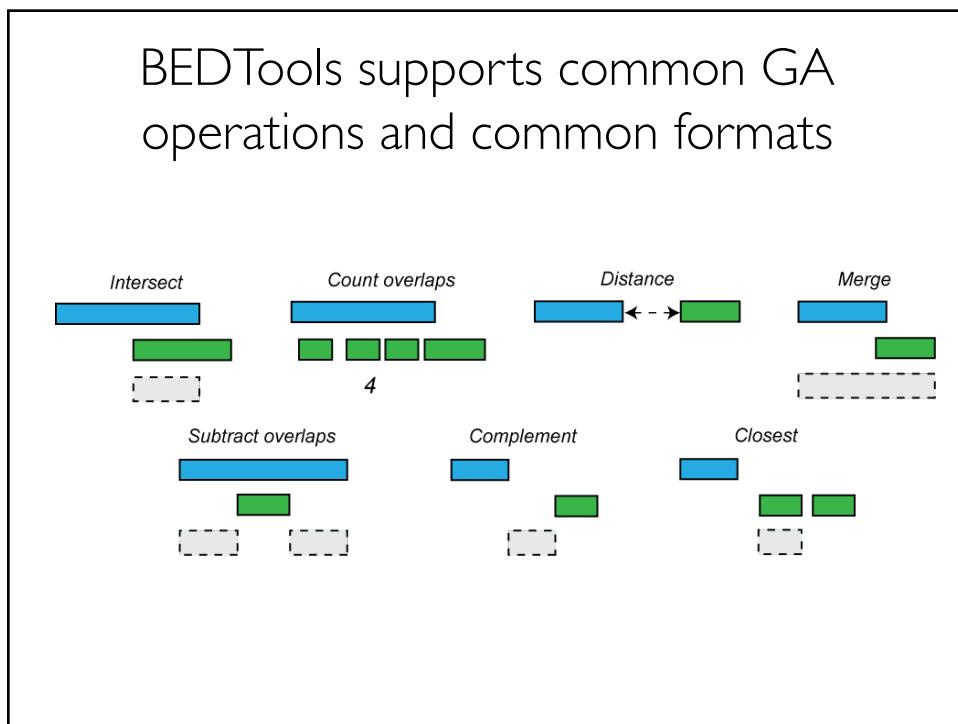
Stable releases for bedtools were formerly archived on Google Code. Unfortunately, the Google Code downloads facility is shutting down; so henceforth, all source code and stable releases will be maintained via this GitHub repository.

Full documentation: <http://bedtools.readthedocs.org>

Summary

Collectively, the bedtools utilities are a swiss-army knife of tools for a wide-range of genomics analysis tasks. The most widely-used tools enable genome arithmetic: that is, set theory on the genome. For example, bedtools allows one to intersect, merge, count, complement, and shuffle genomic intervals from multiple files in widely-used genomic file formats such as BAM, BED, GFF/GTF, VCF.

While each individual tool is designed to do a relatively simple task (e.g., intersect two interval files), quite sophisticated analyses can be conducted by combining multiple bedtools operations on the



BEDTools + UNIX

Real world usage

Find SNPs that have the potential to alter gene expression regulation by affecting methylation at CpG islands.

(restrict to chrom 1)

Step 1: Get annotations

- Single-nucleotide polymorphisms (SNPs)
- CpG islands
- Genes

SNPs from dbSNP via UCSC

The screenshots illustrate the workflow for extracting SNP data from dbSNP via the UCSC Genome Browser:

- Table Browser Interface:** Shows the 'Table Browser' menu bar and various filter options like 'group by', 'genomic tracks', 'assembly', 'db', 'table schema', etc.
- Select Fields Dialog:** Shows the 'Select Fields' dialog for the hg19 genome. Numerous fields are checked, including 'chrom', 'chromStart', 'chromEnd', 'name', 'score', 'strand', 'observed', and 'func'.
- Resulting Table:** Shows the resulting table of SNP data. The columns correspond to the selected fields: #chrom, chromStart, chromEnd, name, score, strand, observed, and func.

Terminal Output:

```
Terminal — bash — 87x26
$ wc -l snps.bed
2064872 snps.bed
```

Output Summary:

"snps.bed"
N = 2,064,872

CpG islands via UCSC

The screenshot shows the UCSC Table Browser interface. In the search bar, the query 'CpG islands' is entered. The results page displays a table with columns: chromosome, chromStart, chromEnd, length, and cgNum. A terminal window to the right shows the command 'grep:B10-280 arq5x4 wc -l cpg.bed' followed by the output '2463'. The terminal also shows the contents of the 'cpg.bed' file.

| chromosome | chromStart | chromEnd | length | cgNum |
|------------|------------|----------|--------|-------|
| chr1 | 28735 | 29810 | 1075 | 116 |
| chr1 | 327793 | 328233 | 440 | 116 |
| chr1 | 327790 | 328229 | 439 | 29 |
| chr1 | 437151 | 438164 | 1013 | 84 |
| chr1 | 533219 | 534114 | 895 | 94 |
| chr1 | 544738 | 546649 | 1911 | 171 |
| chr1 | 762416 | 763445 | 1029 | 66 |
| chr1 | 762416 | 763445 | 1029 | 115 |

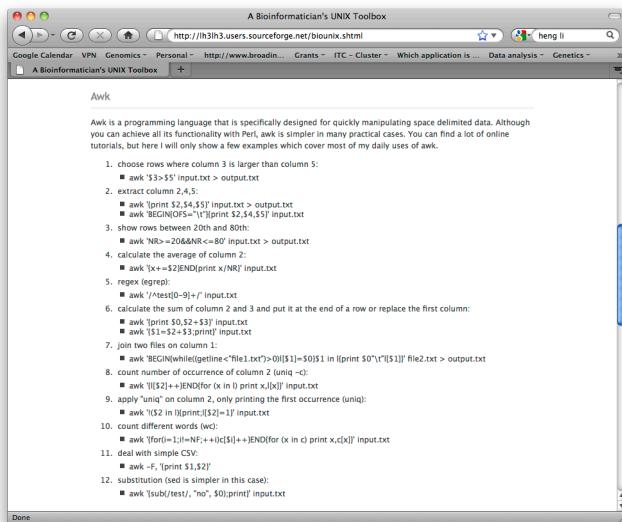
"cpg.bed"
N = 2,463

Problem: No name and no strand in the output. Let's fix it...

The awesome power of **awk**

- written in 1977 by Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan
- Similar constructs as Perl (\$1, \$2, \$3, etc.)
- Typically used for filtering, summarizing, or reorganizing files.
- "One liners", used on files or "streams"
- `awk '{print "Hello World!\n'}'` (Print "HelloWorld!")
- `awk '$2 >= 30' foo.txt` (Get lines in foo where 2nd col. is g.t.e.30)
- `awk '{print $3,$2,$7,$1}' foo.txt > foo.reordered.txt`

The awesome power of **awk**



A screenshot of a web browser window titled "A Bioinformatician's UNIX Toolbox". The page contains a list of awk command examples under the heading "Awk". The commands are numbered and cover various operations like filtering rows, extracting columns, calculating averages, and performing regular expression matches.

```

1. choose rows where column 3 is larger than column 5:
   ■ awk '$3>$5' input.txt > output.txt

2. extract column 2,4,5:
   ■ awk '{print $2,$4,$5}' input.txt > output.txt
   ■ awk BEGIN{NR=1}{print $2,$4,$5} input.txt

3. show rows between 20th and 80th:
   ■ awk 'NR>20&NR<80' input.txt > output.txt

4. calculate the average of column 2:
   ■ awk '{x+= $2}END{print x/NR}' input.txt

5. regex (egrep):
   ■ awk '/^test[0-9]+$/ {print $1}'

6. calculate the sum of columns 2 and 3 and put it at the end of a row or replace the first column:
   ■ awk '{sum=$2+$3} END{for(i=1;i<NR;i++) print sum,$2,$3}' input.txt
   ■ awk '$1=$2+$3' input.txt

7. join two files on column 1:
   ■ awk 'BEGIN{while((getline <"file1.txt">>O){if($1==S){$1="";O=(print $0"\t"$1)" file2.txt > output.txt}}}' S=$1 file1.txt file2.txt

8. count number of occurrence of column 2 (uniq -o):
   ■ awk '{if($2>1){END{for(x in arr) print x,arr[x]}}}' arr[$2]++ input.txt

9. apply "uniq" on column 2, only printing the first occurrence (uniq):
   ■ awk '!C2 in {print $2=1}' input.txt

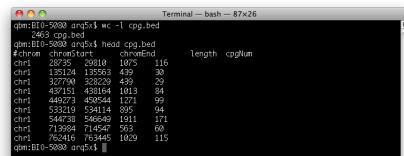
10. count different words (wc):
    ■ awk '{for(i=1;i<NF;i++) c[$i]++}' input.txt
    ■ awk '{for(i=1;i<NF;i++) c[$i]++} END{for(x in c) print x,c[x]}' input.txt

11. deal with simple CSV:
    ■ awk -F, '{print $1,$2}'

12. substitution (sed is simpler in this case):
    ■ awk '{sub(/test/, "no", $0)}'

```

Use awk to fix cpg.bed



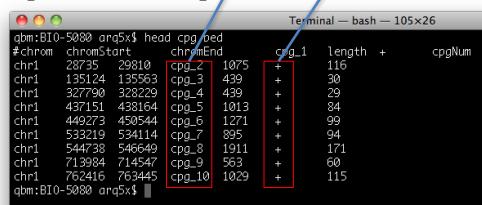
A screenshot of a terminal window titled "Terminal — bash — 87x26". It shows the contents of a file named "cpg.bed" which contains genomic data with columns for chromosome, start position, end position, strand, length, and a numerical value.

| #chrom | chromStart | chromEnd | cpg_1 | length | cpgNum |
|--------|------------|----------|-------|--------|--------|
| chr1 | 28735 | 29810 | 1075 | 116 | |
| chr1 | 135124 | 135563 | 439 | 30 | |
| chr1 | 327790 | 328229 | 439 | 29 | |
| chr1 | 437151 | 438164 | 1013 | 84 | |
| chr1 | 449273 | 450544 | 1271 | 99 | |
| chr1 | 533219 | 534114 | 895 | 94 | |
| chr1 | 544738 | 546649 | 1911 | 171 | |
| chr1 | 713984 | 714547 | 563 | 60 | |
| chr1 | 762416 | 763445 | 1029 | 115 | |

Problem: No name and no strand in the output. Let's fix it..

```
awk '{OFS="\t"; print $1,$2,$3,"cpg_"NR,$4,"+",$5}' cpg.bed > cpg.full.bed
mv cpg.full.bed cpg.bed
```

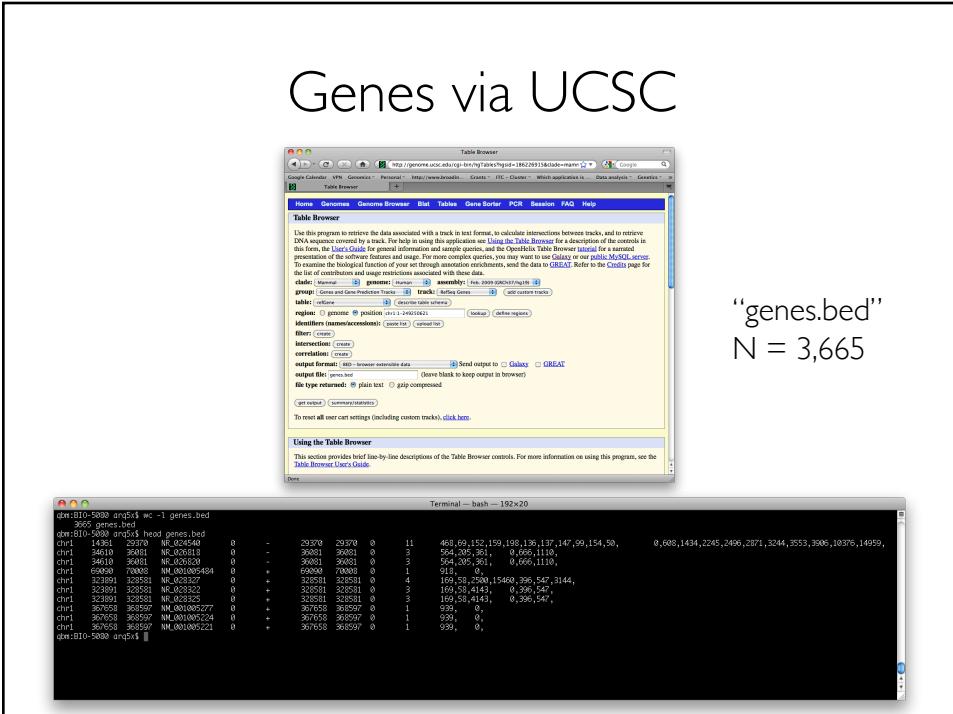
OFS — output field separator; NR — record number



A screenshot of a terminal window titled "Terminal — bash — 105x26". It shows the contents of a file named "cpg.full.bed" which has been modified to include a header and strand information. The columns are labeled "#chrom", "chromStart", "chromEnd", "cpg_1", "length", and "cpgNum". The "cpg_1" column now includes a '+' sign for positive strand and a '-' sign for negative strand.

| #chrom | chromStart | chromEnd | cpg_1 | length | cpgNum |
|--------|------------|----------|------------|--------|--------|
| chr1 | 28735 | 29810 | cpg_-1075 | 116 | |
| chr1 | 135124 | 135563 | cpg_3439 | 30 | |
| chr1 | 327790 | 328229 | cpg_4439 | 29 | |
| chr1 | 437151 | 438164 | cpg_51013 | 84 | |
| chr1 | 449273 | 450544 | cpg_61271 | 99 | |
| chr1 | 533219 | 534114 | cpg_7895 | 94 | |
| chr1 | 544738 | 546649 | cpg_81911 | 171 | |
| chr1 | 713984 | 714547 | cpg_9563 | 60 | |
| chr1 | 762416 | 763445 | cpg_101029 | 115 | |

Genes via UCSC



“genes.bed”
N = 3,665

Find SNPs that have the potential to alter gene expression regulation by affecting methylation at CpG islands.

Let's put it all together by using BEDTools

Step 1: Find SNPs in CpG islands

```
# How many SNPs are there on chrom 1?  
wc -l snps.bed  
2064872  
# Find SNPs that overlap CpG islands using intersectBed  
bedtools intersect -a snps.bed -b cpg.bed -wa >  
snps.cpg.bed  
# How many SNPs overlapped a CpG island?  
wc -l snps.cpg.bed  
14974  
BEDTools typically puts the second file in memory, and reads  
the first file line-by-line, so put the smaller file second.  
-wa report the intersecting -a region,  
normally, only the intersection is reported
```

So, 0.7% of the SNPs were in CpG islands. Is this sensible?

Step 1: Find SNPs in CpG islands

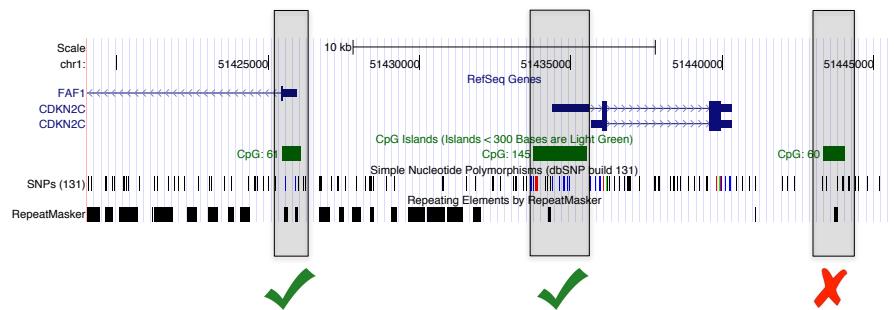
```
# How many SNPs are there on chrom 1?  
wc -l snps.bed  
2064872  
# Find SNPs that overlap CpG islands using intersectBed  
bedtools intersect -a snps.bed -b cpg.bed -wa > snps.cpg.bed  
# How many SNPs overlapped a CpG island?  
wc -l snps.cpg.bed  
14974
```

So, 0.7% of the SNPs were in CpG islands. Is this sensible?

```
# How much of chrom1 do the CpG islands occupy?  
awk '{sum+=$7} END{print sum}' cpg.bed  
1881629
```

$$1,881,629 / 249,250,621 = 0.75\%$$

Step 2: Find TSS/promoter CpG islands



We want to make sure the CpG is near the start of a gene

Step 2: Find TSS/promoter CpG islands

use "window" from BEDTools

By default, bedtools window adds 1000 bp upstream and downstream of each A feature and searches for features in B that overlap this "window". If an overlap is found in B, both the *original* A feature and the *original* B feature are reported (expands window for intersection, produces more columns):

```
$ cat A.bed
chr1 100 200
$ cat B.bed
chr1 500 1000
chr1 1300 2000
$ bedtools window -a A.bed -b B.bed
chr1 100 200 chr1 500 1000
```

```
bedtools window -l 10000 -r 0 -sw -a genes.bed -b cpg.bed | \
cut -f 13-19 | \
sort | \
uniq > cpg.gene_impact.bed
```

-l (add to left coordinate)
-r (add to right coordinate)
-sw (add/subtract based on strand)

Step 3: Find SNPs in TSS/promoter CpG islands
use intersectBed from BEDTools

```
bedtools intersect -a snps.cpg.bed -b  
cpg.gene_impact.bed > snps.cpg.gene_impact.bed
```

Step 3: Find SNPs in TSS/promoter CpG islands
use intersectBed from BEDTools

```
bedtools intersect -a snps.cpg.bed -b  
cpg.gene_impact.bed > snps.cpg.gene_impact.bed
```

What types of SNPs are they? Eg., A→G, G→T

```
grep -v "\-" snps.cpg.gene_impact.bed | \  
cut -f 7 | \  
sort | \  
uniq -c
```

BEDTools

- Genome file types/formats
 - format types
 - BED files
- Overview of genome features
- Genome arithmetic
 - approaches
 - UCSC "binning" algorithm
- Introduction to BEDtools
 - intersection / window / many more

Analysis of transcription regulation

1. Affy-chip/RNA-seq for RNA expression levels
2. Identify differentially expressed genes (edgeR, DESeq2)
 - (possibly) identify subsets of genes associated with different pathways
3. Isolate regions of DNA around (pathway-specific) coordinately expressed (induced) genes
 - BEDtools
4. Take collections of candidate regulatory regions and look for common DNA motif
 - meme, HOMER