

Genomic Regulation

Ivó Hernández, Helena Liz, Diego Fuentes, Fernando Freire

March 3, 2019

Contents

1	Genomic Regulation	2
1.1	Get chr16 CTCF segments	2
1.1.1	Test files	2
1.1.2	Methods	6
1.1.3	Tests	8
1.1.4	Goal	9
1.2	Segment annotation	9
1.2.1	Tracks to annotate	9
1.2.2	Annotate gene overlap	10
1.2.3	Annotate exon overlap	11
1.3	DNASE I overlap	11
1.3.1	Tests	12
1.3.2	Overlap	12

1 Genomic Regulation

1.1 Get chr16 CTCF segments

Get the *chr16* segments which share the same state between both monocyte replicates.

1.1.1 Test files

Script 1.1.1 (text)

```
1 %%bash
2 # Obtain files for test
3 cd RESULTS/Modelo_11_estados
4 cat Monocyte1_11_Master_11_segments.bed | grep 'chr16' | grep 'E9' | sort -k 2,3 -h | head
  → -n 20 > monocyte1_segments.bed
5 cat Monocyte2_11_Master_11_segments.bed | grep 'chr16' | grep 'E9' | sort -k 2,3 -h | head
  → -n 20 > monocyte2_segments.bed
6 echo "Monocyte 1 segments.bed"
7 cat monocyte1_segments.bed
8 wc -l monocyte1_segments.bed
9 echo "Monocyte 2 segments.bed"
10 cat monocyte2_segments.bed
11 wc -l monocyte2_segments.bed
```

Output

```
Monocyte 1 segments.bed
chr16      60400      61400      E9
chr16      72600      72800      E9
chr16     115200     116000      E9
chr16     146400     147400      E9
chr16     156600     157600      E9
chr16     167800     168200      E9
chr16     232200     232400      E9
chr16     412000     412600      E9
chr16     441800     442200      E9
chr16     486400     486800      E9
chr16     537600     538000      E9
chr16     597000     597600      E9
chr16     629000     629400      E9
chr16     661000     661600      E9
chr16     710800     711200      E9
chr16     711600     711800      E9
chr16     736200     736400      E9
chr16     761200     763200      E9
chr16     835400     836200      E9
chr16     1019400    1019600      E9
    20 monocyte1_segments.bed
Monocyte 2 segments.bed
chr16      60400      61400      E9
chr16      72400      72800      E9
chr16     115000     116400      E9
```

chr16	146600	147400	E9
chr16	155400	158200	E9
chr16	167800	168800	E9
chr16	231800	232200	E9
chr16	309000	309200	E9
chr16	353600	354200	E9
chr16	402200	403200	E9
chr16	412000	412800	E9
chr16	441800	442200	E9
chr16	508200	508400	E9
chr16	537400	538200	E9
chr16	596600	597400	E9
chr16	627800	630600	E9
chr16	660800	661800	E9
chr16	710800	711800	E9
chr16	717400	718200	E9
chr16	735800	736800	E9

20 monocyte2_segments.bed

Script 1.1.2 (text)

```

1 %%writefile files/test_tracks/bed1.bed
2 chr16      60400      61400      E9
3 chr16      72600      72800      E9
4 chr16      115200     116000     E9
5 chr16      146400     147400     E9
6 chr16      156600     157600     E9
7 chr16      167800     168200     E9
8 chr16      232200     232400     E9
9 chr16      412000     412600     E9
10 chr16     441800     442200     E9
11 chr16     486400     486800     E9
12 chr16     537600     538000     E9
13 chr16     597000     597600     E9
14 chr16     629000     629400     E9
15 chr16     661000     661600     E9
16 chr16     710800     711200     E9
17 chr16     711600     711800     E9
18 chr16     736200     736400     E9
19 chr16     761200     763200     E9
20 chr16     835400     836200     E9
21 chr16    1019400    1019600     E9

```

Output

Writing files/test_tracks/bed1.bed

Script 1.1.3 (text)

```

1 %%writefile files/test_tracks/bed2.bed
2 chr16      60400      61400      E9
3 chr16      72400      72800      E9
4 chr16      115000     116400     E9
5 chr16      146600     147400     E9
6 chr16      146610     147400     E8
7 chr16      155400     158200     E9
8 chr16      167800     168800     E9
9 chr16      231800     232200     E9
10 chr16      309000     309200     E9
11 chr16      353600     354200     E9
12 chr16      402200     403200     E9
13 chr16      412000     412800     E9
14 chr16      441800     442200     E9
15 chr16      508200     508400     E9
16 chr16      537400     538200     E9
17 chr16      596600     597400     E9
18 chr16      596700     597400     E8
19 chr16      596700     597400     E8
20 chr16      627800     630600     E9
21 chr16      660800     661800     E9
22 chr16      710800     711800     E9
23 chr16      717400     718200     E9
24 chr16      735800     736800     E9

```

Output

Writing files/test_tracks/bed2.bed

Script 1.1.4 (text)

```

1 %%writefile files/test_tracks/dnase1.peaks.bed
2 chr1      770942      771278      chr1.9      584      .      0.039      1.79
   ↳      -1      151
3 chr1      771678      771933      chr1.10     568      .      0.0343     1.5
   ↳ 6      -1      121
4 chr1      773279      773398      chr1.11     555      .      0.0303     1.3
   ↳ 8      -1      49
5 chr1      777497      777598      chr1.12     553      .      0.0299     1.3
   ↳ 6      -1      46
6 chr1      794051      794336      chr1.13     569      .      0.0344     1.5
   ↳ 7      -1      152
7 chr1      800514      800667      chr1.14     549      .      0.0287
   ↳ 1.3     -1      34
8 chr1      805004      805656      chr1.15     1000     .      0.3561     16
   ↳      -1      286
9 chr16     63392      63462      chr16.1     551      .      0.0292     1.33
   ↳      -1      26

```

10	chr16	65192	65500	chr16.2	650	.	0.0582	2.69	」
	↪	-1	140						
11	chr16	65680	65848	chr16.3	578	.	0.0371		
	↪ 1.7	-1	81						
12	chr16	66552	66704	chr16.4	565	.	0.0334	1.52	」
	↪	-1	71						
13	chr16	69567	69918	chr16.5	593	.	0.0416	1.91	」
	↪	-1	156						
14	chr16	72620	73427	chr16.6	1000	.	0.2652	12.	」
	↪ 5	-1	256						
15	chr16	74047	74486	chr16.7	687	.	0.069		
	↪ 3.2	-1	213						
16	chr16	77159	77214	chr16.8	550	.	0.0289	1.31	」
	↪	-1	20						
17	chr16	78558	80270	chr16.9	756	.	0.0889	4.15	」
	↪	-1	1188						
18	chr16	80539	84109	chr16.10	797	.	0.101	4.72	」
	↪	-1	2835						
19	chr16	84632	86116	chr16.11	1000	.	0.2262	10	」
	↪ .7	-1	648						
20	chr16	86223	86873	chr16.12	717	.	0.0777	3.6	」
	↪ 2	-1	274						
21	chr16	87530	87809	chr16.13	614	.	0.0475	2.1	」
	↪ 9	-1	147						
22	chr16	88254	89597	chr16.14	703	.	0.0736	3.4	」
	↪ 2	-1	691						
23	chr16	91933	92367	chr16.15	558	.	0.0312	1.4	」
	↪ 2	-1	78						
24	chr16	102933	104351	chr16.16	1000	.	0.521	1	」
	↪ 6	-1	852						
25	chr16	107650	109038	chr16.17	708	.	0.0751	3	」
	↪ .49	-1	327						
26	chr16	109773	109913	chr16.18	555	.	0.0305	1	」
	↪ .39	-1	61						
27	chr16	110337	110607	chr16.19	581	.	0.0381	1	」
	↪ .74	-1	117						
28	chr16	110941	114948	chr16.20	915	.	0.1354	6	」
	↪ .36	-1	421						
29	chr16	115448	116225	chr16.21	1000	.	0.3633		」
	↪ 16	-1	419						
30	chr16	116533	116836	chr16.22	568	.	0.0343	1	」
	↪ .56	-1	131						

Output

Writing files/test_tracks/dnase1.peaks.bed

1.1.2 Methods

Script 1.1.5 (python)

```
1 import re as re
2
3 def head(path, filename, lines=20):
4     """
5     """
6     i = 0
7     file = open(path + "/" + filename, "r")
8     for line in file:
9         print(line.strip())
10        i += 1
11        if i > lines:
12            break
13    file.close()
14
15 def get_parts(line, sep='\t'):
16     """
17     """
18    bed_line_parts = line.rstrip('\n').split(sep)
19    return bed_line_parts[0], int(bed_line_parts[1]), int(bed_line_parts[2]),
20    ↪ bed_line_parts[3]
21
22 def concat_parts(chrom, start, end, feature, sep='\t'):
23     """
24     """
25    bed_line = chrom + '\t' + str(start) + '\t' + str(end) + '\t' + feature + '\n'
26    return bed_line
27
28 def bed_coverage(path, filename, sep='\t'):
29     """
30     Returns the accumulated length of all the segments of the bed file filename
31     """
32    i = 0
33    file = open(path + "/" + filename, "r")
34    coverage = 0
35    for line in file:
36        _, f1_segment_start, f1_segment_end, _ = get_parts(line)
37        coverage += f1_segment_end - f1_segment_start
38    file.close()
39    return coverage
40
41 def bed_segment_count_by_re_feature(path, filename, re_feature, sep='\t'):
42     """
43     Returns the segment count by feature name of the bed file filename.
44     The feature is informed as a regexp
45     """
46    i = 0
47    file = open(path + "/" + filename, "r")
48    segment_count = 0
49    for line in file:
```

```

49         _, _, _, segment_feature = get_parts(line)
50         if re.search(re_feature, segment_feature):
51             segment_count += 1
52     file.close()
53     return segment_count
54
55 def intersect_bed(input_dir, input_file1, input_file2, output_dir, output_file, chrom="chr16"
56 ",
57                 f1_feature_filter="E9", f2_feature_filter="E9", output_feature="E9",
58                 ↪ sep='\t',
59                 drop_feature_threshold=20, output_mode="intersect"):
60     """
61     If output mode is intersect, returns the intersected bed segments
62     If output mode is annotate, returns all the segments of input_file1
63     annotated if it's the case with the feature defined in input_file2.
64     """
65     f1_segments = open(input_dir + "/" + input_file1, "r")
66     f2_segments = open(input_dir + "/" + input_file2, "r")
67     output_segments = open(output_dir + "/" + output_file, "w")
68     f1_segment = f1_segments.readline()
69     f2_segment = f2_segments.readline()
70     while(f1_segment != "" and f2_segment != ""):
71         f1_chrom, f1_segment_start, f1_segment_end, f1_feature = get_parts(f1_segment)
72         f2_chrom, f2_segment_start, f2_segment_end, f2_feature = get_parts(f2_segment)
73         feature = f1_feature
74         # Filter f1 and read f1
75         if f1_chrom != chrom or (f1_feature_filter != "" and f1_feature != f1_feature_filter)
76 ):
77             f1_segment = f1_segments.readline()
78             # Filter f2 and read f2
79             elif f2_chrom != chrom or (f2_feature_filter != "" and f2_feature !=
80             ↪ f2_feature_filter):
81                 f2_segment = f2_segments.readline()
82                 # f2 segment downstream f1 segment
83                 elif f2_segment_start > f1_segment_end:
84                     if output_mode == "annotate" and drop_feature_threshold < f1_segment_end -
85                     ↪ f1_segment_start:
86                         output_segment = concat_parts(chrom, f1_segment_start, f1_segment_end,
87                         ↪ feature)
88                         output_segments.write(output_segment)
89                         f1_segment = f1_segments.readline()
90                         # f1 segment downstream f2 segment
91                         elif f1_segment_start > f2_segment_end:
92                             f2_segment = f2_segments.readline()
93                         else: # Intersect
94                             # Save intersect
95                             if output_mode == "intersect":
96                                 output_start = max(f1_segment_start, f2_segment_start)
97                                 output_end = min(f2_segment_end, f1_segment_end)
98                                 if drop_feature_threshold < output_end - output_start:
99                                     output_segment = concat_parts(chrom, output_start, output_end,
100                                     ↪ output_feature)

```

```

94         output_segments.write(output_segment)
95     else:
96         feature = f1_feature + "+" + output_feature
97         # Advance f1
98         if f2_segment_end >= f1_segment_end:
99             if output_mode == "annotate" and drop_feature_threshold < f1_segment_end -
100                 ↪ f1_segment_start:
101                 output_segment = concat_parts(chrom, f1_segment_start, f1_segment_end,
102                 ↪ feature)
103                 output_segments.write(output_segment)
104                 f1_segment = f1_segments.readline()
105                 # Advance f2
106                 elif f1_segment_end > f2_segment_end:
107                     f2_segment = f2_segments.readline()
108                     while(output_mode == "annotate" and f1_segment != ""):
109                         output_segments.write(f1_segment)
110                         f1_segment = f1_segments.readline()
111
112 f1_segments.close()
113 f2_segments.close()
114 output_segments.close()

```

1.1.3 Tests

Script 1.1.6 (python)

```

1  PATH = "files/tracks"
2  TEST_PATH = "files/test_tracks"
3  M1_FILE = "Monocyte1_11_Master_11_segments.bed"
4  M2_FILE = "Monocyte2_11_Master_11_segments.bed"
5  M1_FILE_TEST = "bed1.bed"
6  M2_FILE_TEST = "bed2.bed"
7  SEP = '\t'
8  CHROM = "chr16"
9  STATE = "E9"
10
11 intersect_bed(TEST_PATH, M1_FILE_TEST, M2_FILE_TEST, TEST_PATH,
12               STATE + "_segments_test.bed", chrom=CHROM,
13               f1_feature_filter=STATE, f2_feature_filter=STATE, output_feature=STATE,
14               ↪ sep=SEP,
15               drop_feature_threshold=300)
16 head(STATE + "_segments_test.bed", 20)

```

Output

chr16	60400	61400	E9
chr16	115200	116000	E9
chr16	146600	147400	E9
chr16	156600	157600	E9

chr16	167800	168200	E9
chr16	412000	412600	E9
chr16	441800	442200	E9
chr16	537600	538000	E9
chr16	597000	597400	E9
chr16	629000	629400	E9
chr16	661000	661600	E9
chr16	710800	711200	E9

1.1.4 Goal

Script 1.1.7 (python)

```

1 intersect_bed(PATH, M1_FILE, M2_FILE, PATH, STATE + "_segments.bed", chrom=CHROM,
2               f1_feature_filter = STATE, f2_feature_filter = STATE, output_feature =
3               ↪ STATE, sep=SEP,
4               drop_feature_threshold = 10)
5 head(STATE + "_segments.bed", 10)

```

Output

chr16	60400	61400	E9
chr16	72600	72800	E9
chr16	115200	116000	E9
chr16	146600	147400	E9
chr16	156600	157600	E9
chr16	167800	168200	E9
chr16	412000	412600	E9
chr16	441800	442200	E9
chr16	537600	538000	E9
chr16	597000	597400	E9
chr16	629000	629400	E9

1.2 Segment annotation

Annotate the segments. At a minimum, the percentage of segments that overlap with protein-coding genes in said chromosome should be given.

1.2.1 Tracks to annotate

The tracks are obtained from <https://genome.ucsc.edu/cgi-bin/hgTables>

1.2.2 Annotate gene overlap

Script 1.2.1 (python)

```
1 OVERLAP_TYPE = "_gene_overlap"
2 OUTPUT_FEATURE = STATE + OVERLAP_TYPE
3
4 intersect_bed(PATH, STATE + "_segments.bed", "hg19_genes.bed",
5             PATH, OUTPUT_FEATURE + "_annotate.bed", chrom = CHROM,
6             f1_feature_filter = STATE, f2_feature_filter="",
7             output_feature = OUTPUT_FEATURE, sep=SEP,
8             drop_feature_threshold = 10, output_mode="annotate")
9 head(PATH, OUTPUT_FEATURE + "_annotate.bed", 10)
10
11 overlap_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FEATURE +
12     ↪ "_annotate.bed",
13                                     OUTPUT_FEATURE)
14 print("")
15 print("Count of state segments overlapped:", overlap_segment_count)
16 total_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FEATURE +
17     ↪ "_annotate.bed", "")
18 print("Count of all state segments", total_segment_count)
19 print("Percent overlapped state segments over total segments:",
20     overlap_segment_count * 100 / total_segment_count)
```

Output

chr16	60400	61400	E9
chr16	72600	72800	E9
chr16	115200	116000	E9+E9_gene_overlap
chr16	146600	147400	E9+E9_gene_overlap
chr16	156600	157600	E9+E9_gene_overlap
chr16	167800	168200	E9+E9_gene_overlap
chr16	412000	412600	E9
chr16	441800	442200	E9+E9_gene_overlap
chr16	537600	538000	E9+E9_gene_overlap
chr16	597000	597400	E9+E9_gene_overlap
chr16	629000	629400	E9

Count of state segments overlapped: 173

Count of all state segments 468

Percent overlapped state segments over total segments: 36.965811965811966

1.2.3 Annotate exon overlap

Script 1.2.2 (python)

```
1 OVERLAP_TYPE = "_exon_overlap"
2 OUTPUT_FEATURE = STATE + OVERLAP_TYPE
3
4 intersect_bed(PATH, STATE + "_segments.bed", "hg19_coding_exons.bed",
5             PATH, OUTPUT_FEATURE + "_annotate.bed", chrom = CHROM,
6             f1_feature_filter = STATE, f2_feature_filter="",
7             output_feature = OUTPUT_FEATURE, sep=SEP,
8             drop_feature_threshold = 10, output_mode="annotate")
9 head(PATH, OUTPUT_FEATURE + "_annotate.bed", 10)
10
11 overlap_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FEATURE +
12     ↪ "_annotate.bed",
13                                     OUTPUT_FEATURE)
14 print("")
15 print("Count of state segments overlapped:", overlap_segment_count)
16 total_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FEATURE +
17     ↪ "_annotate.bed", "")
18 print("Count of all state segments", total_segment_count)
19 print("Percent overlapped state segments over total segments:",
20     overlap_segment_count * 100 / total_segment_count)
```

Output

chr16	60400	61400	E9
chr16	72600	72800	E9
chr16	115200	116000	E9
chr16	146600	147400	E9
chr16	156600	157600	E9
chr16	167800	168200	E9
chr16	412000	412600	E9
chr16	441800	442200	E9
chr16	537600	538000	E9
chr16	597000	597400	E9+E9_exon_overlap
chr16	629000	629400	E9

Count of state segments overlapped: 11

Count of all state segments 468

Percent overlapped state segments over total segments: 2.3504273504273505

1.3 DNase I overlap

Download the peaks of DNase I in monocytes of ENCODE for chr16 and calculate the percentage of overlap between DNaseI-peaks and your work segments. Use the file wgEncodeOpenChromDnaseMonocd14Pk.narrowPeak.gz in: <http://hgdownload.cse.ucsc.edu/goldenpath/hg19/encodeDCC/wgEncodeOpenChromDnase>

1.3.1 Tests

Script 1.3.1 (python)

```
1 intersect_bed(TEST_PATH, STATE + "_segments_test.bed", "dnase1.peaks.bed", TEST_PATH,
2               STATE + "_dnase1_overlap_test.bed",
3               chrom = CHROM, f1_feature_filter = STATE, f2_feature_filter = "",
4               output_feature = STATE + "_dnase1_overlap", sep=SEP,
5               drop_feature_threshold=20)
6 head(TEST_PATH, STATE + "_dnase1_overlap_test.bed", 10)
```

Output

chr16	115448	116000	E9_dnase1_overlap
-------	--------	--------	-------------------

1.3.2 Overlap

Overlap by coverage Overlap calculated as percent ratio between sum of base pair overlapped and sum of total base pair covered by all the E9 segments. This method doesn't have much sense because it depends of the arbitrary sensitivity of the dna base segments (200 bps in the case of chromatin states).

Script 1.3.2 (python)

```
1 intersect_bed(PATH, STATE + "_segments.bed",
2               ↪ "wgEncodeOpenChromDnaseMonocd14Pk.narrowPeak.bed",
3               PATH, STATE + "_dnase1_overlap.bed", chrom = CHROM,
4               f1_feature_filter = STATE, f2_feature_filter = "",
5               output_feature = STATE + "_dnase1_overlap", sep = SEP,
6               drop_feature_threshold = 10)
7 head(PATH, "E9_dnase1_overlap.bed", 10)
8 coverage_peaks = bed_coverage(PATH, "wgEncodeOpenChromDnaseMonocd14Pk.narrowPeak.bed",
9                               ↪ sep='\t')
10 coverage_state = bed_coverage(PATH, STATE + "_segments.bed", sep='\t')
11 print("")
12 print("Coverage DNASE peaks:", coverage_peaks, "bps")
13 print("Coverage E9:", coverage_state, "bps")
14 print("Percent overlap over total coverage peaks:", coverage_state * 100 / coverage_peaks)
```

Output

chr16	72620	72800	E9_dnase1_overlap
chr16	115448	116000	E9_dnase1_overlap
chr16	146819	147400	E9_dnase1_overlap
chr16	157056	157367	E9_dnase1_overlap
chr16	167800	168118	E9_dnase1_overlap
chr16	412000	412600	E9_dnase1_overlap
chr16	441800	442200	E9_dnase1_overlap
chr16	537761	538000	E9_dnase1_overlap
chr16	597000	597400	E9_dnase1_overlap
chr16	629000	629400	E9_dnase1_overlap

```
chr16      661000      661548      E9_dnase1_overlap

Coverage DNASE peaks: 22653113851288 bps
Coverage E9: 43366167200 bps
Percent overlap over total coverage peaks: 0.19143578884866774
```

Overlap by segment count Overlap calculated as percent ratio between segment count of overlapped E9-DNASE segments and total count of E9 segments.

Script 1.3.3 (python)

```
1 OUTPUT_FEATURE = STATE + "_dnase1_overlap"
2
3 intersect_bed(PATH, STATE + "_segments.bed",
4   ↳ "wgEncodeOpenChromDnaseMonocd14Pk.narrowPeak.bed",
5       PATH, STATE + "_dnase1_overlap_annotate.bed", chrom = CHROM,
6       f1_feature_filter = STATE, f2_feature_filter="",
7       output_feature = OUTPUT_FEATURE, sep=SEP,
8       drop_feature_threshold = 10, output_mode="annotate")
9 head(PATH, STATE + "_dnase1_overlap_annotate.bed", 10)
10
11 overlap_segment_count = bed_segment_count_by_re_feature(PATH, STATE +
12   ↳ "_dnase1_overlap_annotate.bed",
13                                     OUTPUT_FEATURE)
14 print("")
15 print("Count of state segments overlapped:", overlap_segment_count)
16 total_segment_count = bed_segment_count_by_re_feature(PATH, STATE +
17   ↳ "_dnase1_overlap_annotate.bed", "")
18 print("Count of all state segments", total_segment_count)
19 print("Percent overlapped state segments over total segments:",
20       overlap_segment_count * 100 / total_segment_count)
```

Output

```
chr16      60400      61400      E9
chr16      72600      72800      E9+E9_dnase1_overlap
chr16      115200     116000     E9+E9_dnase1_overlap
chr16      146600     147400     E9+E9_dnase1_overlap
chr16      156600     157600     E9
chr16      167800     168200     E9
chr16      412000     412600     E9+E9_dnase1_overlap
chr16      441800     442200     E9+E9_dnase1_overlap
chr16      537600     538000     E9+E9_dnase1_overlap
chr16      597000     597400     E9+E9_dnase1_overlap
chr16      629000     629400     E9+E9_dnase1_overlap
```

```
Count of state segments overlapped: 127
Count of all state segments 468
Percent overlapped state segments over total segments: 27.136752136752136
```

Automated verifications

Script 1.3.4 (python)

```
1 segment_count_annotate = bed_segment_count_by_re_feature(PATH, STATE +  
  ⇨ "_dnase1_overlap_annotate.bed", "")  
2 segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_segments.bed", "")  
3 assert segment_count_annotate == segment_count,\  
4     "Count of annotated segments not equal to count of original segments"
```

Visual inspection

Script 1.3.5 (text)

```
1 %%bash  
2 export TRACKS=files/tracks/  
3 echo "Counts"  
4 wc -l ${TRACKS}E9_dnase1_overlap_annotate.bed  
5 wc -l ${TRACKS}E9_segments.bed  
6 tail ${TRACKS}E9_dnase1_overlap_annotate.bed  
7 echo  
8 tail ${TRACKS}E9_segments.bed  
9 echo  
10 head ${TRACKS}E9_dnase1_overlap_annotate.bed  
11 echo  
12 head ${TRACKS}E9_segments.bed  
13 echo  
14 echo "Counts of annotations:"  
15 cat ${TRACKS}E9_dnase1_overlap_annotate.bed | grep "E9_dnase1_overlap" | wc -l  
16 cat ${TRACKS}E9_segments.bed | grep "" | wc -l
```

Output

Counts

468 files/tracks/E9_dnase1_overlap_annotate.bed

468 files/tracks/E9_segments.bed

chr16	89233600	89234800	E9
chr16	89527000	89527400	E9
chr16	89623800	89624200	E9+E9_dnase1_overlap
chr16	89707800	89708000	E9+E9_dnase1_overlap
chr16	89772400	89772600	E9+E9_dnase1_overlap
chr16	89927000	89927800	E9
chr16	89976600	89977000	E9+E9_dnase1_overlap
chr16	90092400	90092800	E9+E9_dnase1_overlap
chr16	90182400	90183000	E9
chr16	90281600	90282000	E9
chr16	89233600	89234800	E9
chr16	89527000	89527400	E9
chr16	89623800	89624200	E9
chr16	89707800	89708000	E9
chr16	89772400	89772600	E9

chr16	89927000	89927800	E9
chr16	89976600	89977000	E9
chr16	90092400	90092800	E9
chr16	90182400	90183000	E9
chr16	90281600	90282000	E9
chr16	60400	61400	E9
chr16	72600	72800	E9+E9_dnase1_overlap
chr16	115200	116000	E9+E9_dnase1_overlap
chr16	146600	147400	E9+E9_dnase1_overlap
chr16	156600	157600	E9
chr16	167800	168200	E9
chr16	412000	412600	E9+E9_dnase1_overlap
chr16	441800	442200	E9+E9_dnase1_overlap
chr16	537600	538000	E9+E9_dnase1_overlap
chr16	597000	597400	E9+E9_dnase1_overlap
chr16	60400	61400	E9
chr16	72600	72800	E9
chr16	115200	116000	E9
chr16	146600	147400	E9
chr16	156600	157600	E9
chr16	167800	168200	E9
chr16	412000	412600	E9
chr16	441800	442200	E9
chr16	537600	538000	E9
chr16	597000	597400	E9

Counts of annotations:

127

468