**Using bedtools in a linux environment**

**1. Logging in and testing some things**

**1.1 Mac users:** start **terminal** (typically in utilities->terminal). In the terminal:

ssh -X your\_ku\_user\_id@ricco.popgen.dk

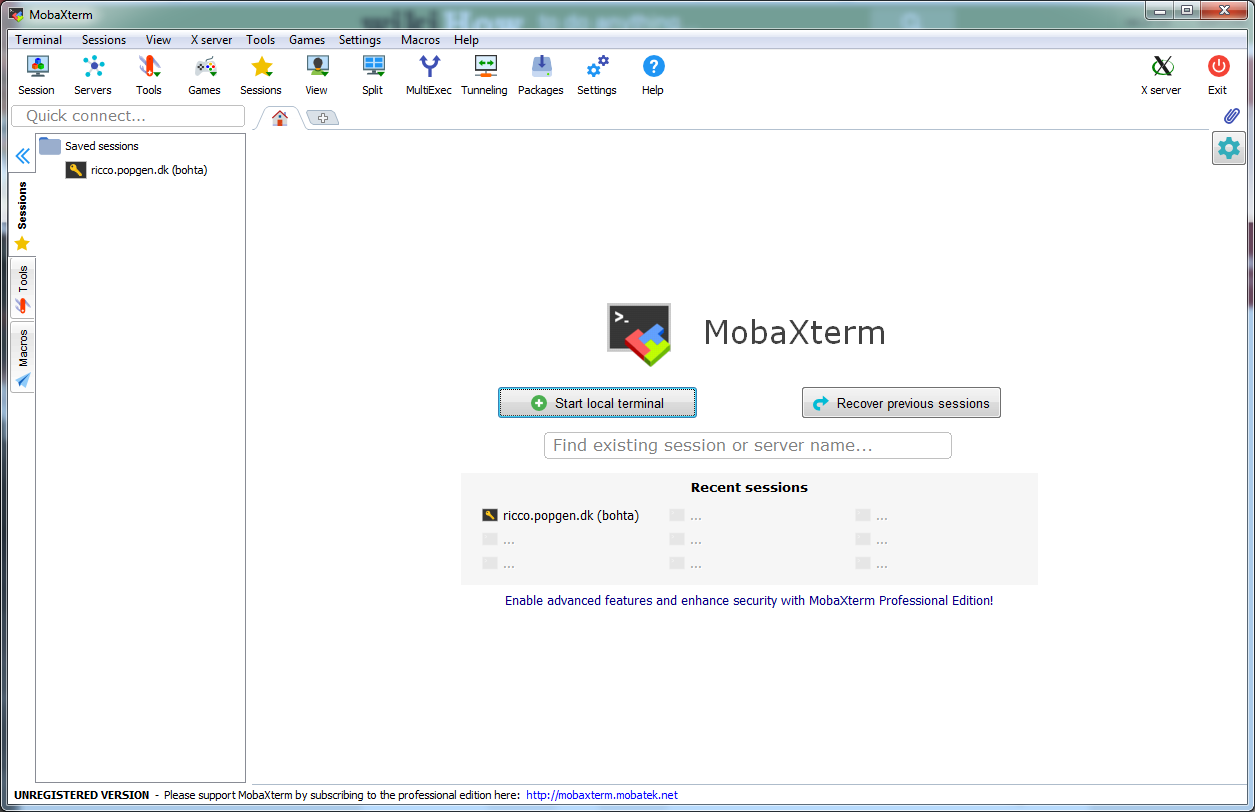
password: bohta2017

**1.2: Other unix/linux users**: same as mac users, but terminal is probably already active.

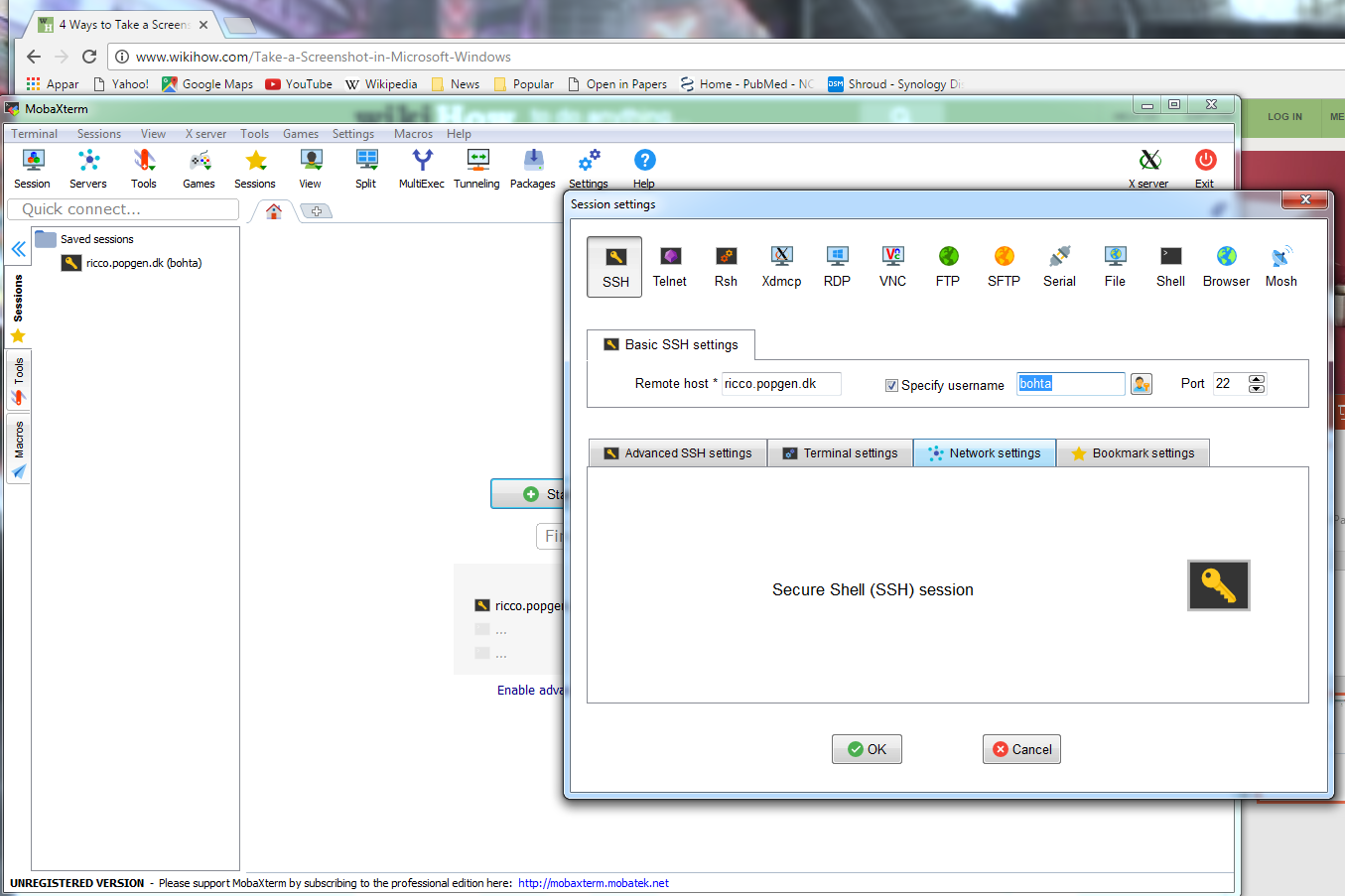
**1.3 Windows users:** Download and install mobaxterm (free edition)

<http://mobaxterm.mobatek.net/download.html>

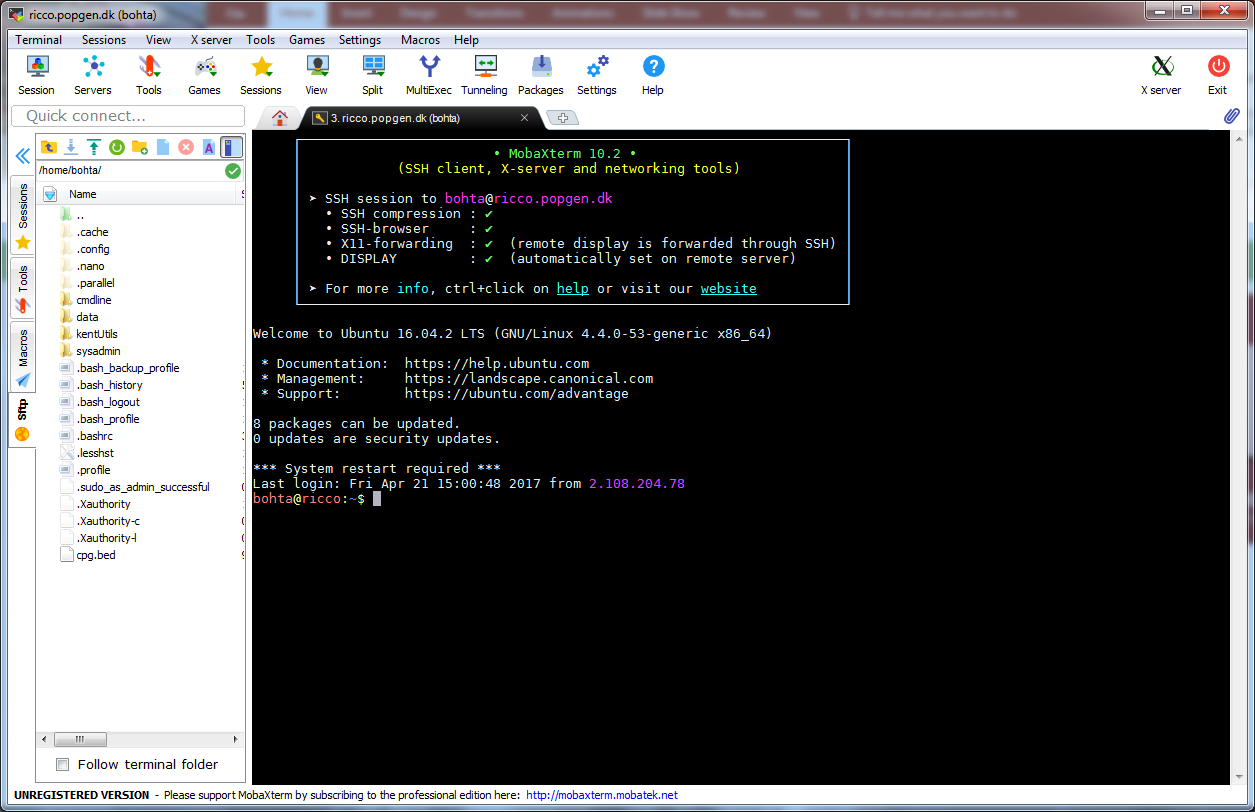
In windows, open MobaXterm, and start a new ssh session



Fill in the form as below. If you save this as a session you will not have to fill this more than once.



Your login should now look something like this:



Unix and mac users should see the same, but only the “black” part.

**1.4 Orientation:** You are now in your “home directory”. In the examples below I am logging into the teacher account, called bohta. This means I start in the /home/bohta directory. This will change depending on what your user name is.

Last login: Mon Apr 17 21:48:59 2017 from 2.108.204.78

**bohta@ricco:~$**

You can test this by typing pwd

**bohta@ricco:~$** pwd

/home/bohta

First, make a data directory called “data”. We will use this for the bedtools exercises below.

mkdir data

use ls to see that you created a new directory called data

Now, move into the data directory by

cd data

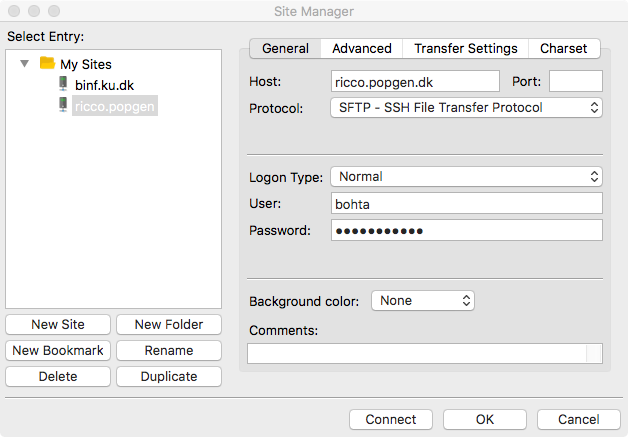
We will now populate this directory with some useful .bed and .wig files for the exercises. This assumes you are now within your data folder

cp /home/bohta/data/\* ./

This copies everything in /home/bohta/data into the current folder (./)

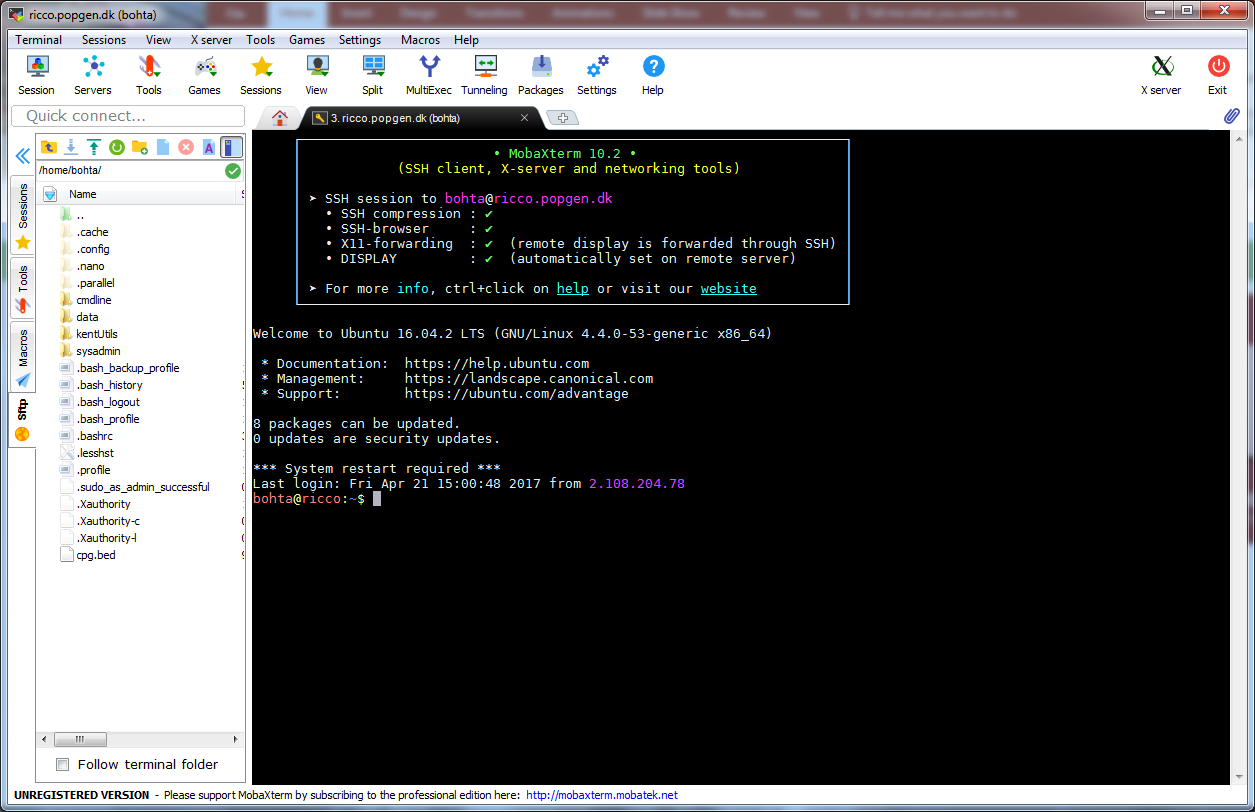
**2. Transferring files between your local computer and the linux server and vice versa.** In later excercises, we will transfer data to and from the linux server to your own computer. This is how to do it:

Mac/lLnux: Download the FileZilla program (free). Click File->Site manager. Here, fill in what si below – you can also save the setting as a “site” so that you will not have to fill in in all the time like I have done. Change User to your ku id.



This will give a quite simple view where you computer’s file system is to the left and the linux system is to the right. These can be used to transfer files either way by just dragging.

Windows: Windows users are lucky in the MobAXterm also supports sFTP. The field left of your terminal (once logged in) has the file structure of your linux account (see picture bwkow) The easiest way to move files from your computer is just dragging and dropping files into or from this. It effectively works like the sftp program above.



**3. Bedtools** tutorial (based on <http://quinlanlab.org/tutorials/bedtools/bedtools.html>)

It is important to try out the commands below yourself while reading. In particular, look at the files you may be producing with head or less.

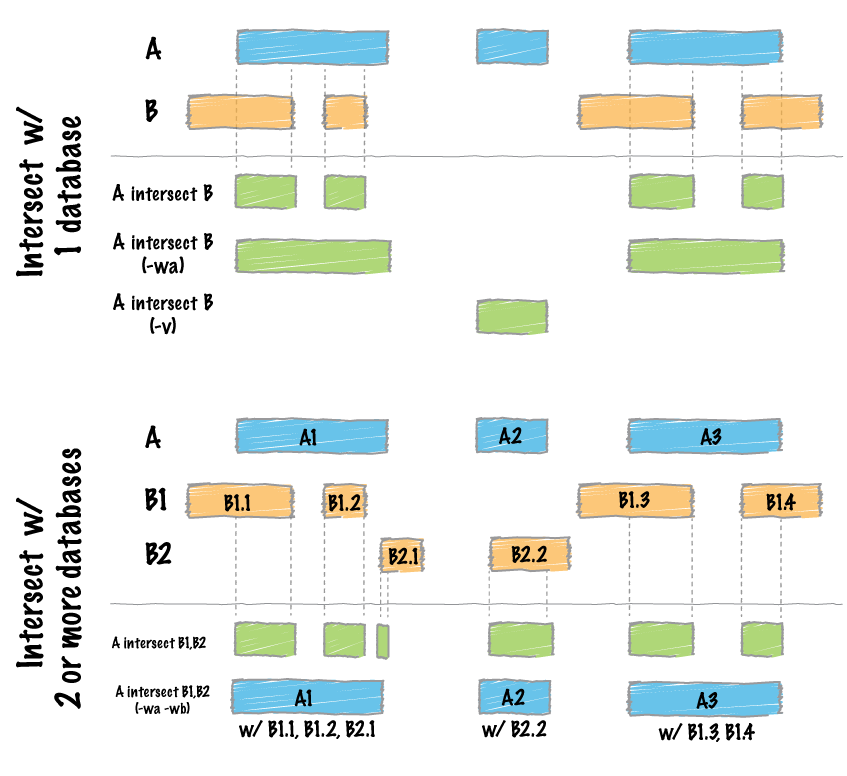
The intersect command is the workhorse of the bedtools suite. It compares two or more BED/BAM/VCF/GFF files and identifies all the regions in the genome where the features in the two files overlap (that is, share at least one base pair in common).

Bedtools intersect works by giving it two files, corresponding to two bed files

- a and -b like this:

bedtools intersect -a cpg.bed -b exons.bed

This will investigate the intersection between the regions of cpg.bed and exons.bed. Specifically, it will ask what cpg regions that overlap exons.



By default, intersect reports the intervals that represent overlaps between your two files. Let’s run this command and pipe it through “head” so that we only see the first 5 lines of the intersect.

bedtools intersect -a cpg.bed -b refseq\_exons.bed | head -5

chr1 155188536 155188616 CpG:\_361

chr1 155188536 155188616 CpG:\_361

chr1 155191907 155192004 CpG:\_361

chr1 155191907 155192004 CpG:\_361

chr1 2228694 2229734 CpG:\_366

Each line shows a part of a CpG island that overlapped an exon. The last columns hows the name of the cpg island that overlapped an exon. There are some important features here.

First, by default, only the part of the CpG island that overlapped an exon is shown, not the whole CpG island. This is why the same CpG island name show up four times.

Second, you may have noticed that the first region showed up twice. This is due to that it overlaps two DIFFERENT exons: we don't see it here because we only see the CpGs.

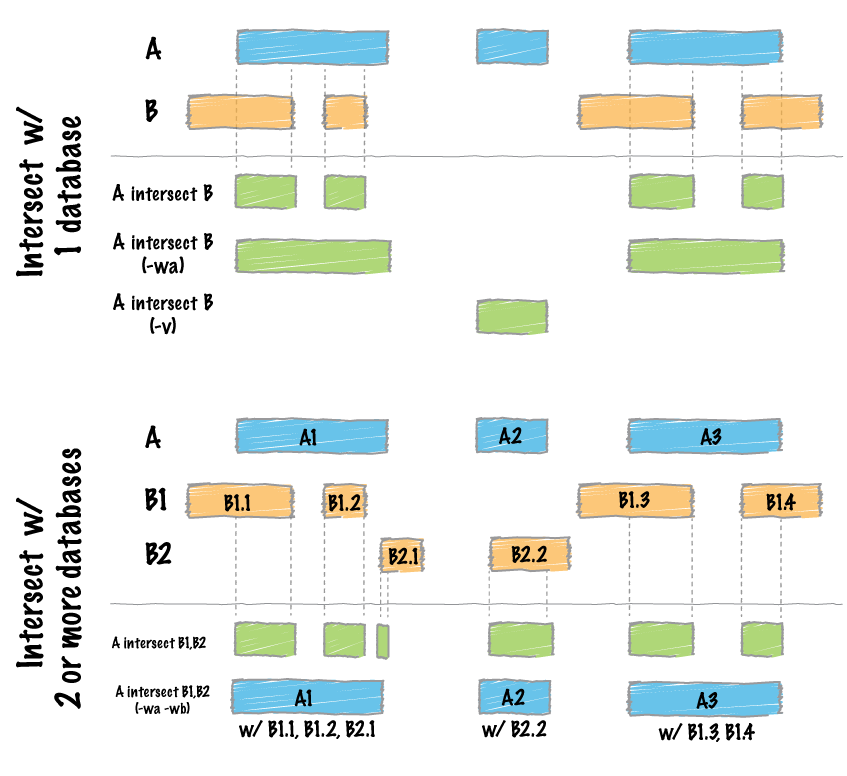
We can save the output as a new bedfile by redirection (the > sign).

bedtools intersect -a cpg.bed -b refseq\_exons.bed >cpg\_exon\_intersect.bed

If you now do an ls, you will see an cpg\_exon\_intersect.bed file. This you can read with, for instance, less

## Reporting the original feature in each file.

The -wa (write A) and -wb (write B) options allow one to see the original records from the A and B files that overlapped. As such, instead of not only showing you where the intersections occurred, it shows you what intersected.



Here we do BOTH wb and wa

bedtools intersect -a cpg.bed -b refseq\_exons.bed -wa -wb | head -5

chr1 155188536 155192004 CpG:\_361 chr1 155188162 155188616 NM\_001204286.1\_exon\_5\_0\_chr1\_155188163\_r 0 -

chr1 155188536 155192004 CpG:\_361 chr1 155188162 155188616 NM\_001204285.1\_exon\_5\_0\_chr1\_155188163\_r 0 -

chr1 155188536 155192004 CpG:\_361 chr1 155191907 155192310 NM\_001204286.1\_exon\_6\_0\_chr1\_155191908\_r 0 -

chr1 155188536 155192004 CpG:\_361 chr1 155191907 155192283 NM\_001204285.1\_exon\_6\_0\_chr1\_155191908\_r 0 -

chr1 2226773 2229734 CpG:\_366 chr1 2228694 2229735 NM\_003036.3\_exon\_0\_0\_chr1\_2228695\_f 0 +

Each row here shows first the coordinates where cpg island and exon overlaps, then the id and coordinates of the whole CpG islands, then the name and coordinate sof the overlapping exon

Here we can see more clearly that indeed, the first cpG islands overlaps two identical exons that are from two differnet transcripts

**How many base pairs of overlap were there?**

The -wo (write overlap) option allows one to also report the *number* of base pairs of overlap between the features that overlap between each of the files. The last column that comes out will show the bp overlap.

bedtools intersect -a cpg.bed -b refseq\_exons.bed -wo | head -10

chr1 155188536 155192004 CpG:\_361 chr1 155188162 155188616 NM\_001204286.1\_exon\_5\_0\_chr1\_155188163\_r 0 - 80

chr1 155188536 155192004 CpG:\_361 chr1 155188162 155188616 NM\_001204285.1\_exon\_5\_0\_chr1\_155188163\_r 0 - 80

chr1 155188536 155192004 CpG:\_361 chr1 155191907 155192310 NM\_001204286.1\_exon\_6\_0\_chr1\_155191908\_r 0 - 97

chr1 155188536 155192004 CpG:\_361 chr1 155191907 155192283 NM\_001204285.1\_exon\_6\_0\_chr1\_155191908\_r 0 - 97

chr1 2226773 2229734 CpG:\_366 chr1 2228694 2229735 NM\_003036.3\_exon\_0\_0\_chr1\_2228695\_f 0 + 1040

chr1 2226773 2229734 CpG:\_366 chr1 2228724 2229735 XM\_005244775.3\_exon\_0\_0\_chr1\_2228725\_f 0 + 1010

chr1 36306229 36307408 CpG:\_110 chr1 36306347 36306424 XM\_011542153.2\_exon\_0\_0\_chr1\_36306348\_f 0 + 77

chr1 36306229 36307408 CpG:\_110 chr1 36306597 36306755 XM\_011542153.2\_exon\_1\_0\_chr1\_36306598\_f 0 + 158

chr1 36306229 36307408 CpG:\_110 chr1 36306841 36306905 XM\_011542153.2\_exon\_2\_0\_chr1\_36306842\_f 0 + 64

chr1 36306229 36307408 CpG:\_110 chr1 36306347 36306424 XM\_017002343.1\_exon\_0\_0\_chr1\_36306348\_f 0 + 77

**Counting the number of overlapping features.**

As we figured out now, each CpG islands can potentially overlap many exons. The count of overlaps may be useful to know. To count for each feature in the “a” file, the number of overlapping features in the “b” file, use the-c option.

bedtools intersect -a cpg.bed -b refseq\_exons.bed -c | head

chr1 155188536 155192004 CpG:\_361 4

chr1 2226773 2229734 CpG:\_366 2

chr1 36306229 36307408 CpG:\_110 34

chr1 47708822 47710847 CpG:\_164 0

chr1 53737729 53739637 CpG:\_221 10

chr1 144179071 144179313 CpG:\_20 0

chr1 156499825 156501268 CpG:\_112 8

chr1 228457406 228458733 CpG:\_120 2

chr1 247857028 247857950 CpG:\_99 1

chr1 28735 29737 CpG:\_111 1

Here we see that the first CpG island overlap four exons.

This is quite useful especially when regions in one set is much wider than in the other sets (like genes vs CpG islands)

Challenge: What is the distribution of CpG islands overlapping genes (not necessarily exons – find the appropriate bed file)? In other words, how many Cpgs have0, 1, 2 et exons overlapping them? This requires a transfer to your computer and some R work (remember, R can read bed files like the above)

**Finding features that DO NOT overlap**

Sometimes we want to identify those features in our A file that **do not** overlap features in the B file. The -v option is your friend in this case.

bedtools intersect -a cpg.bed -b refseq\_whole\_gene.bed -v | head

chr1 47708822 47710847 CpG:\_164

chr1 144179071 144179313 CpG:\_20

chr1 368792 370063 CpG:\_99

chr1 381172 382185 CpG:\_84

chr1 597839 598734 CpG:\_94

chr1 866595 866958 CpG:\_24

chr1 869818 870248 CpG:\_50

chr1 908919 910503 CpG:\_154

chr1 933952 934964 CpG:\_92

chr1 936259 936492 CpG:\_20

**Require a minimal fraction of overlap.**

The intersect default is to report overlaps between features in A and B so long as *at least one basepair* of overlap exists. This is sometimes not the best solution. The -f option allows you to specify what fraction of each feature in A should be overlapped by a feature in B before it is reported.

Let’s be more strict and require 50% of overlap:

bedtools intersect -a cpg.bed -b refseq\_exons.bed -wo -f 0.50

37841 416251 3973213 |head

chr1 247857028 247857950 CpG:\_99 chr1 247857198 247857666 NM\_015431.3\_exon\_0\_0\_chr1\_247857199\_f 0 + 468

chr1 135124 135563 CpG:\_30 chr1 134772 139696 NR\_039983.2\_exon\_0\_0\_chr1\_134773\_r 0 - 439

chr1 491107 491546 CpG:\_29 chr1 490755 494898 NR\_028322.1\_exon\_0\_0\_chr1\_490756\_r 0 - 439

chr1 609358 611269 CpG:\_171 chr1 609082 610750 XM\_011542538.1\_exon\_4\_0\_chr1\_609083\_r 0 - 1392

chr1 778604 779167 CpG:\_60 chr1 778759 779092 XR\_001737594.1\_exon\_0\_0\_chr1\_778760\_f 0 + 333

chr1 778604 779167 CpG:\_60 chr1 778762 779092 XR\_001737595.1\_exon\_0\_0\_chr1\_778763\_f 0 + 330

chr1 778604 779167 CpG:\_60 chr1 778763 779092 XR\_946806.2\_exon\_0\_0\_chr1\_778764\_f 0 + 329

chr1 778604 779167 CpG:\_60 chr1 778764 779092 XR\_001737596.1\_exon\_0\_0\_chr1\_778765\_f 0 + 328

chr1 778604 779167 CpG:\_60 chr1 778808 779092 XR\_001737593.1\_exon\_0\_0\_chr1\_778809\_f 0 + 284

chr1 853483 853831 CpG:\_28 chr1 853390 859446 NR\_047524.1\_exon\_3\_0\_chr1\_853391\_f 0 + 348

Challenge: using unix tools only (no R), compare the number of CpG islands that do not overlap exons when requiring 1 bp overlap compared to 50% overlap? Hint: to count the number of rows in a file: use the wc command

**Merging bed files**

There are many instances where bed regions in the same file overlap each other. This could be genes, ChIP sites, etc. At times, we may want to merge overlapping regions into a single regions. The merge tool can do this for us.

An important issue is that merge merge requires that the bed regions are sorted (in other words, that the regions follow each other in the file). Luckily, the exon.bed file is already sorted. Should you have a bed file that is not sorted, one can use the linux sort command to sort it (see the end of this tutorial on how to do it)

Merging results in a new set of intervals representing the merged set of intervals in the input. That is, if a base pair in the genome is covered by 10 features, it will now only be represented once in the output file. Luckily, the exon.bed file is already sorted, so we do have to sort it.

There is a program within linux that can do the sorting for us - appropriately called sort.

If I have a bed file called foo.bed, I can sort it (and save the results in a new file) by saying

sort –k1,1 –k2,2n foo.bed > sorted\_foo.bed

The -k1,1 etc syntax tells sort what columns in the bed files to do the sorting with.

For instance, if you have an input file **data.txt** with the following data:

01 Joe

02 Marie

03 Albert

04 Dave

...and you sort it without any options, like this:

sort data.txt

you will get

01 Joe

02 Marie

03 Albert

04 Dave

which is exactly the same as we started with – this is due to that sort starts by looking at the first lette or number in each row, and those are actually already sorted. If we want to sort based on the names in the second column, we say

sort –k 2 data.txt

This will sort on column 2, giving

03 Albert

04 Dave

01 Joe

02 Marie

What does k -1,1 mean in that case? The second 1 means that the we ONLY sort on the word in the second columns – by default, it will continue until the end of the row (this does not matter in the toy example, but will matter in a bed file, because it has many more columns and characters in each row). We can sort on multiple columns as well.

Challenge:

1) So what does

sort –k1,1 –k2,2n foo.bed > sorted\_foo.bed

actually sort on? Hint: look at what the columns in a bed file are.

2. Make a sorted refseq exon file based on the command above, called sorted\_refseq\_exons.bed

Now we can merge the file:

bedtools merge -i sorted\_refseq\_exons.bed | head -n 5

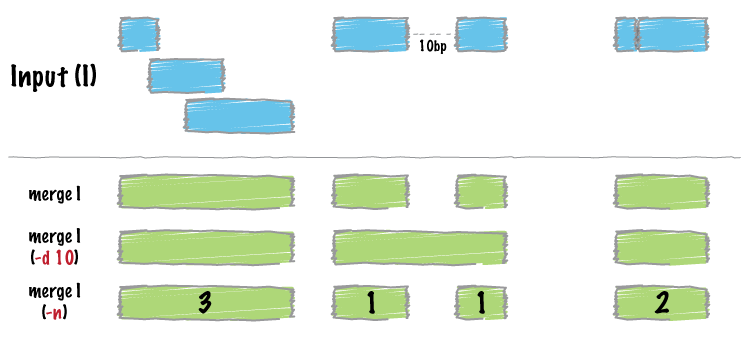
chr1 11873 12227

chr1 12612 12721

chr1 13220 14829

chr1 14969 15038

chr1 15795 15947

What is essentially happening is shown in the figure below, in the green first row. As you see, it is also possible to merge not only overlapping exons but also exons that are close to one anther (with the –d flag). 

A more sophisticated approach would be to not only merge overlapping intervals, but also report the *number* of intervals that were integrated into the new, merged interval – essentially saying how many exons that contributed.

This is done with the -c and -o options. The -c option allows one to specify a column or columns in the input that you wish to summarize. The -o option defines the operation(s) that you wish to apply to each column listed for the -c option. For example, to count the number of overlapping intervals that led to each of the new “merged” intervals, one will “count” the first column (though the second, third, fourth, etc. would work just fine as well).

bedtools merge -i sorted\_refseq\_exons.bed -c 1 -o count | head -n 5

chr1 11873 12227 1

chr1 12612 12721 1

chr1 13220 14829 2

chr1 14969 15038 1

chr1 15795 15947 1

Here we can see that the region is a merge of two exons. Compare this to the original sorted bed file to see which two that were merged.

To this end: what if you wanted to have the actual names of the overlapped regions that made up each merged region? We can do this with the collapse operation available via the -o argument. The name of the exon is in the fourth column, so we ask merge to create a list of the exon names with -c 4 -o collapse:

bedtools merge -i sorted\_refseq\_exons.bed -c 1,4 -o count,collapse | head -5

chr1 11873 12227 1 NR\_046018.2\_exon\_0\_0\_chr1\_11874\_f

chr1 12612 12721 1 NR\_046018.2\_exon\_1\_0\_chr1\_12613\_f

chr1 13220 14829 2 NR\_046018.2\_exon\_2\_0\_chr1\_13221\_f,NR\_024540.1\_exon\_0\_0\_chr1\_14362\_r

chr1 14969 15038 1 NR\_024540.1\_exon\_1\_0\_chr1\_14970\_r

chr1 15795 15947 1 NR\_024540.1\_exon\_2\_0\_chr1\_15796\_r

Again, we can see that the region is a merge of two exons, and now we have their names.

**Genome coverage**

For many analyses, one wants to measure the genome wide coverage of a feature (bed) file. For example, we often want to know what fraction of the genome is covered by 1 feature, 2 features, 3 features, etc. This is relevant when making full-genome sequencing, but also many other genomics technique, like CAGE, RNA-seq or ChIP. This is done with the genomecov tool.

As an example, let’s produce a histogram of coverage of the exons throughout the genome. Like the merge tool, genomecov requires pre-sorted data. It also needs a genome file, which essentially just is a small text file that shows how long the chromosomes are. You have one in your directory called genome.txt (please have a look at it with less)

genomecov takes two mandatory inputs: -i , which is the bed file of interest and –g for the genome file.

genoemcov can give two type of outputs: first, a simple report on how much a dataset is covering each chromosome, and second a type of coverage barplot across the genome.

Let’s make the simple version first. Because this take a lot of computational resources, we are putting a “nice” before the command. This allows for sharing multiple jobs per cpu without crashing the server. If we give the dafult comman, we wil see what fraction of nucleotdies that are covered by 0, 1, 2 etc exons – this is a bit overkill so we say –max 1, which means that all nucleotdies that have 1 or more exons covering will be counted as “1”

nice bedtools genomecov -i sorted\_refseq\_exons.bed -g genome.txt –max 1

chr1 0 237126183 248956422 0.952481

chr1 1 11830239 248956422 0.0475193

chr10 0 128332579 133797422 0.959156

chr10 1 5464843 133797422 0.0408442

chr10\_GL383545v1\_alt 0 165802 179254 0.924956

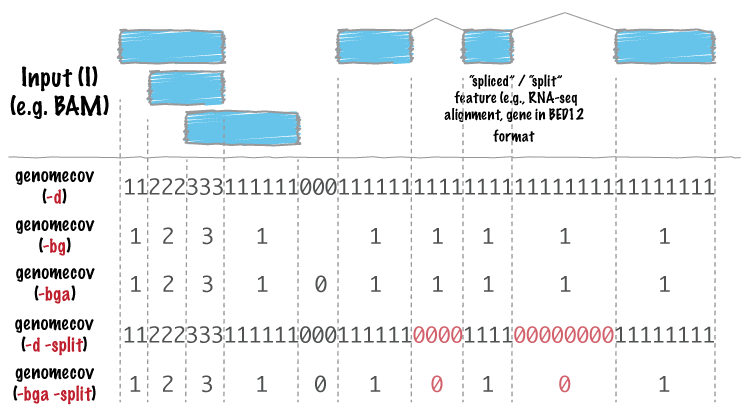
etc.

This shows per column

1. chromosome
2. depth of coverage from features in input file (here 0 means no overlap, 1 means 1 or more)
3. number of bases on chromosome (or genome) with depth equal to column 2.
4. size of chromosome in base pairs
5. fraction of bases on chromosome (or entire genome) with depth equal to column 2.

We can see that chr1 and 10 has a fairly equal fraction of exon coverage – 4-5%. We also see some weird chromosomes including chr10\_GL383545v1\_alt. What are those?

Next, by using the –bg option, we will instead get a bedgraph file output – this essentially is a barplot saying how many exons that overlap every single nucleotide. useful for looking at things in the genome browser as well as other tools. The –d option does the same but outputs a more extended format



The below command takes a few minutes to run. Because of this, we put a “nice” in front of the command, which makes sharing of the server CPUs easier

nice bedtools genomecov -i sorted\_refseq\_exons.bed -g genome.txt -bg | head -20

chr1 11873 12227 1

chr1 12612 12721 1

chr1 13220 14361 1

chr1 14361 14409 2

chr1 14409 14829 1

chr1 14969 15038 1

chr1 15795 15947 1

chr1 16606 16765 1

chr1 16857 17055 1

chr1 17232 17436 1

chr1 17605 17742 1

chr1 17914 18061 1

chr1 18267 18366 1

chr1 24737 24891 1

chr1 29320 29370 1

chr1 29925 30039 1

chr1 30365 30503 1

chr1 30563 30667 1

chr1 30975 31295 1

chr1 34610 35174 1

This is similar to the merge command, but works on a bp level: the file essentially is a barplot on the genome saying how many exons that overlap each bp. The same type of command could for instance be used to say what genomic nucleotides that has the most ChIP signal, methylation, etc. This type of analyses is what underlies profile plots, that we used another tool for.

**4. Very short GALAXY tutorial**

Galaxy is a graphics “front-end” to many tool including bedtools. It is quire straightforward to understand once you have used the linux tools. The best way of learning it is to watch the two videos below, and then explore the system and see the similarities to the linux system.

A nice introduction on how to get data into galaxy using the genome browser is here

<https://vimeo.com/185523444>

The following video shows how to intersect datasets and more, following the last video.

<https://vimeo.com/185538367>

Here, they use Galaxys own “intersect” tools that are similar to bedtools, but it serves the purpose of understanding the basic operations in galaxy. However, if you look at the tool set available in Galaxy, bedtools is also there, and they are recommended to use instead. As you see, the same types of analysis that you can do in the command line you can also do with the graphical tool.

