Ramón Tamarit Agusti

FUNDAMENTOS DE INFORMÁTICA EN ENTORNOS BIOINFORMÁTICOS

PEC1 — Primera Prueba de evaluación continua

Ejercicio 1 – Descripción de los catálogos de genes. Análisis comparativo de los genomas de varias especies (catálogos de genes disponibles a día de hoy).

Ejercicio 1 – Descripción de los catálogos de genes [30%]

Análisis comparativo de los genomas de varias especies (catalogos de genes disponibles a día de hoy). De la misma forma que hemos realizado durante los ejercicios prácticos, conectaos al servidor UCSC para acceder a los ficheros "refGene.txt" de varias especies. El objetivo es que relleneis la siguiente Tabla con los datos que obtendreis usando los comandos apropiados en vuestro terminal de LINUX. Añadid una pequeña interpretación biológica de los resultados anotados:

Species	# Transcripts	# Genes	Average Length Transcripts	Average # exons per transcript
H. sapiens				
M. musculus				
G. gallus				
D. melanogaster				
C. elegans				

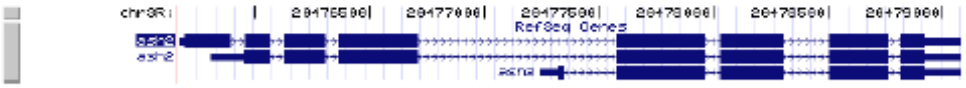
NOTA: El símbolo "#" denota "número de" (cardinal)

Recordad que la estructura de estos ficheros es una línea por tránscrito (RNA mensajero, RNAm) y que un gen puede dar lugar a más de un transcrito alternativo (isoforma).

Por ejemplo, este es un extracto del correspondiente fichero en *D.melanogaster*:

```
741     NM_170159       chr3R   +       20475664        20479099        20475692 20478924 8
20475664,20475956,20476126,20476371,20477582,20478031,20478503,20478819,
20475899,20476066,20476311,20476716,20477963,20478312,20478761,20479099,  15170
ash2 cmpl       cmpl    0,0,2,1,1,1,0,0,

741     NM_176558       chr3R   +       20475809        20479099        20475950 20478924 7
20475809,20476126,20476371,20477582,20478031,20478503,20478819,
20476066,20476311,20476716,20477963,20478312,20478761,20479099,    15171
ash2    cmpl cmpl 0,2,1,1,1,0,0,

741     NM_170160       chr3R   +       20477248        20479098        20477324 20478924 5
20477248,20477582,20478031,20478503,20478819,
20477352,20477963,20478312,20478761,20479098,       15172
ash2    cmpl    cmpl 0,1,1,1,0,0,
```

Donde hay 3 RNAm (cada uno con su identificador NM) para un mismo gen (*ash2*). Esto significa que este gen *ash2* tiene 3 formas alternativas de transcripción. Gráficamente, este es el gen y sus tres isoformas (con sus exones):



Ficheros refGene.txt:

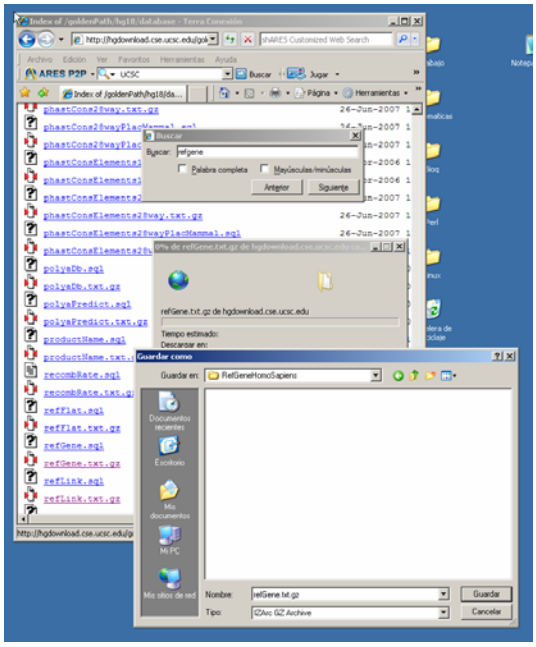
- UCSC web: http://hgdownload.cse.ucsc.edu/downloads.html

[seleccionad la especie y buscar el directorio "Annotation database"]

1. Paso 1. Preaparición y Obtención de los ficheros.

Para trabajar los datos usare el terminal de bash de cygwin. Previamente he instalado las ultimas versiones de los módulos de gawk. Como editor de texto usaré indistintamente el notepad2 (http://www.flos-freeware.ch/notepad2.html) y xemacs. La ejecución de todos estos comandos en cygwin es equivalente a hacerlo en un Terminal "Linux" o "Unix", con la salvedad que el entorno grafico nos permite observar con facilidad la estructura de los ficheros para analizarlos.

Una vez obtenidos los ficheros del servidor de UCSC (en fecha 18.11.2008), los descomprimo (*gunzip nombrefichero*) los ficheros en cada una de sus carpetas y los renombro de la siguiente forma. (Uso *cp nombrei nombref* para copiar y renombrar a la vez)

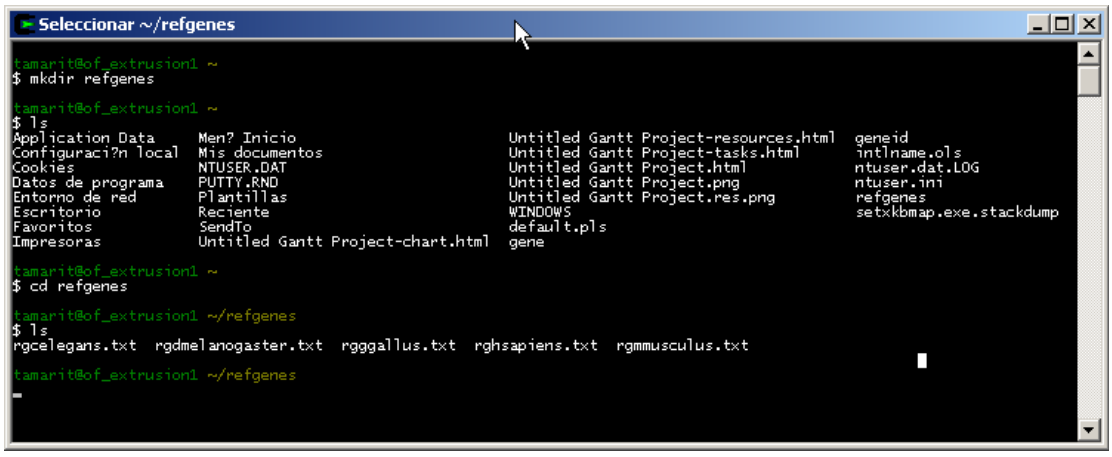


Vamos al servidor de UCSC y descargamos los ficheros de anotaciones de cada especie. Cada uno de los ficheros lo guardo en un directorio para que no se mezclen.

- H. sapiens rghsapiens.txt
- M. musculus rgmmusculus.txt
- G. gallus rggallus.txt
- D. melanogaster rgdmelanogaster.txt
- C. elegans rgcelegans.txt

Los ficheros txt son los que usaré en el terminal de Unix.

El directorio de trabajo sera `~/refgenes` , dentro de el creo otro directorio /cpy y pongo otra copia sobre la que trabajaré. (ver figura de abajo)



Estructura de directorios después de descomprimir, renombrar los ficheros y traspasarlos al directorio de trabajo de Unix

2. Estructura de los ficheros. RefGene.txt.

Del servidor de UCSC se puede consultar la estructura de la tabla RefGene. La estructura la detallo a continuación:

field	example	Description
bin	1643	Indexing field to speed chromosome range queries.
name	NM_016459	Name of gene (usually transcript_id from GTF)
chrom	chr5	Reference sequence chromosome or scaffold
strand	-	+ or - for strand
txStart	138751155	Transcription start position
txEnd	138753504	Transcription end position
cdsStart	138751352	Coding region start
cdsEnd	138753444	Coding region end
exonCount	4	Number of exons
exonStarts	138751155,138751608,1387520...	Exon start positions
exonEnds	138751509,138751719,1387521...	Exon end positions
id	0	Unique identifier
name2	MGC29506	Alternate name (e.g. gene_id from GTF)
cdsStartStat	cmpl	enum('none','unk','incmpl','cmpl')
cdsEndStat	cmpl	enum('none','unk','incmpl','cmpl')
exonFrames	2,2,0,0,	Exon frame {0,1,2}, or -1 if no frame for exon

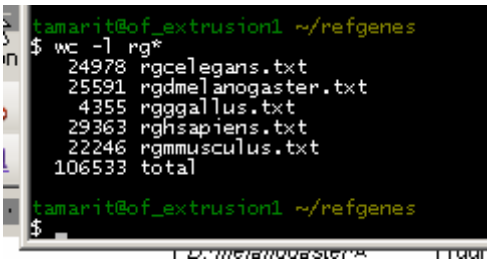
Los campos que nos interesan son:

- Campo 5 “txStart” . Inicio del trascrito
- Campo 6 “txEnd”, final del trascrito
- Campo 9 “exonCount” : Número de exones
- Campo 12 “name2” : Nombre del gen

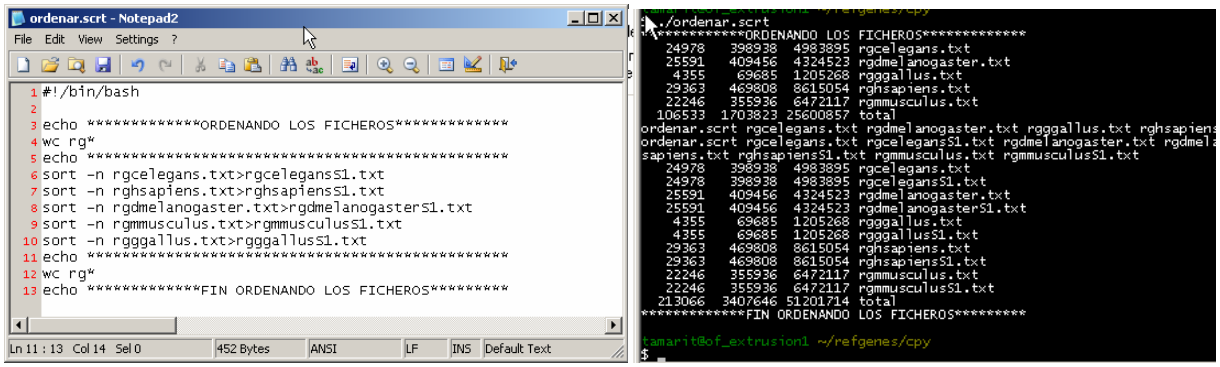
3. Cálculo del número de trascritos.

El número de transcritos se puede obtener del conteo de las filas de cada fichero. Para ello usaré directamente el comando `wc rg*`. La primera columna es el número de líneas (transcritos). Igualmente se puede ejecutar alternativamente el comando `wc -l rg*` que nos dará únicamente el número de líneas:

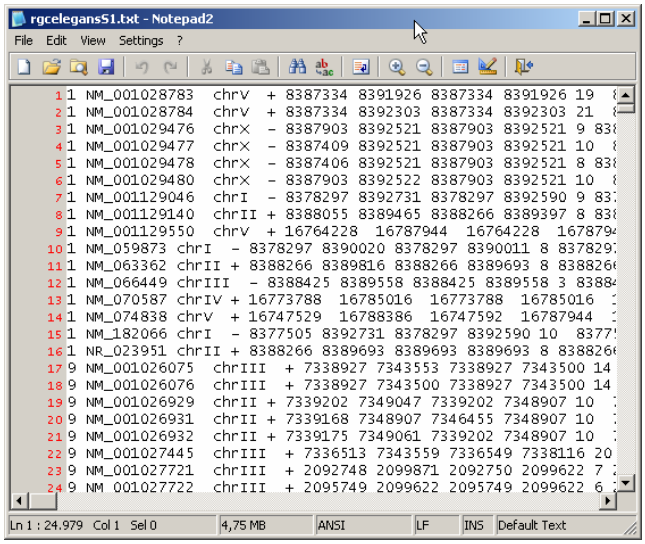
```
tamarit@of_extrusion1 ~/refgenes
$ wc rg*
   24978   398938  4983895 rgcelegans.txt
   25591   409456  4324523 rgdmelanogaster.txt
    4355    69685  1205268 rggallus.txt
   29363   469808  8615054 rghsapiens.txt
   22246   355936  6472117 rgmmusculus.txt
  106533  1703823 25600857 total
```



Vamos a ordenarlos para estudiar el fichero. Para ello voy a ir preparando un script que puede serme útil en el futuro. En principio a partir de ahora trabajare sobre los ficheros de `~/refgenes/cpy`.



Y el resultado:



El tamaño del transcrito será el campo 6 menos el 5

```
$ gawk '{print $13,$6-$5}' rgcelegansS1.txt > test.txt
```

El numero de exones esta en la posición 9

```
$ gawk '{print $13,$6-$5,$9}' rgcelegansS1.txt > test.txt
```

El nombre del gen único esta en el campo 13

4. Contando los genes de cada especie

Vamos a contar los genes. Se identifican por el campo 13. Partimos de las copias y de los ficheros ordenados por el script, es decir los *S1.txt

Ejemplo:

```
$ gawk '{print $13}' rgcelegansS1.txt > rgcelegansGenes.txt
```

Luego ordenamos con sort –u (para eliminar los duplicados). Con wc contamos las ocurrencias.

Al final el script queda:

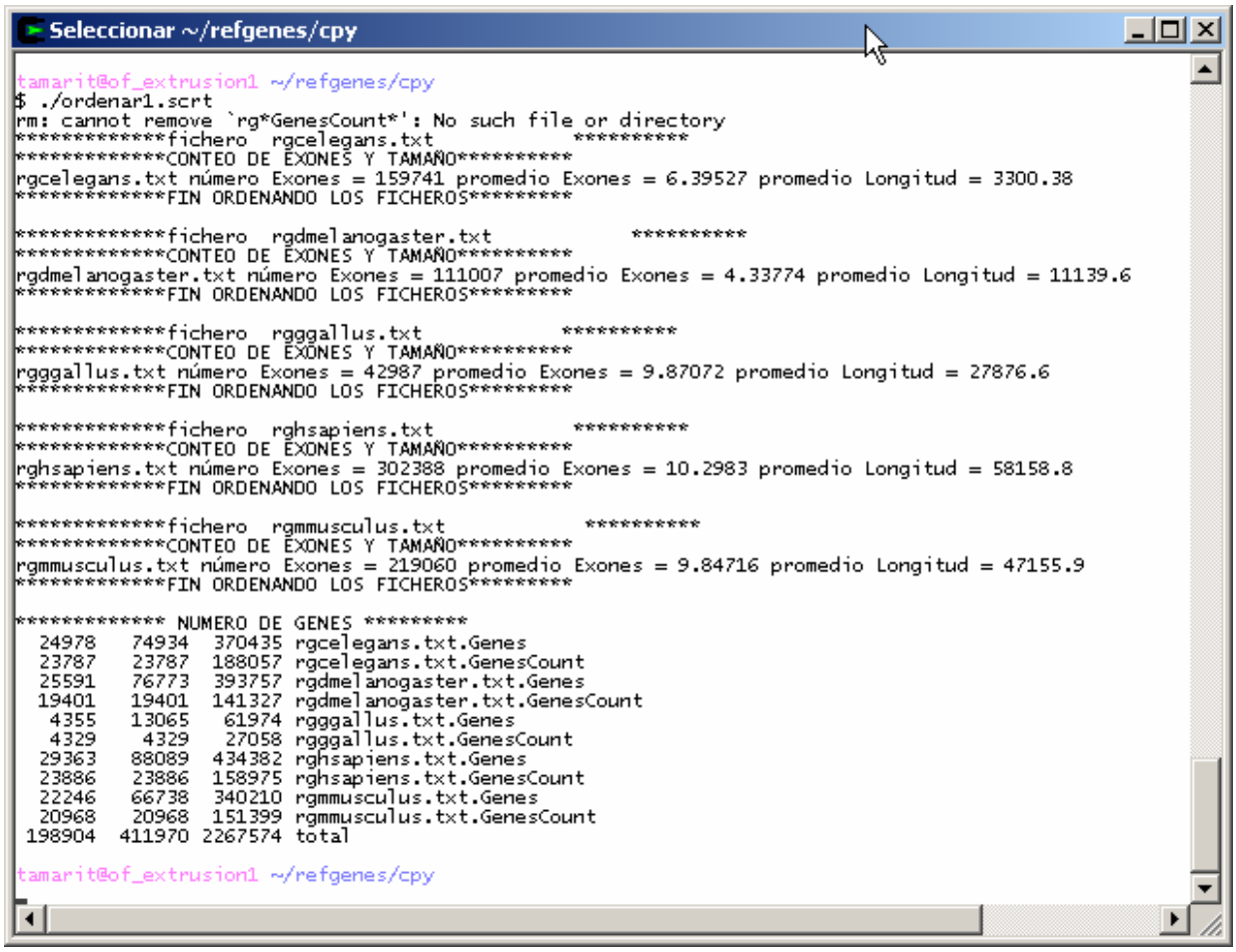
ordenar1.scrt - Notepad2

File Edit View Settings ?

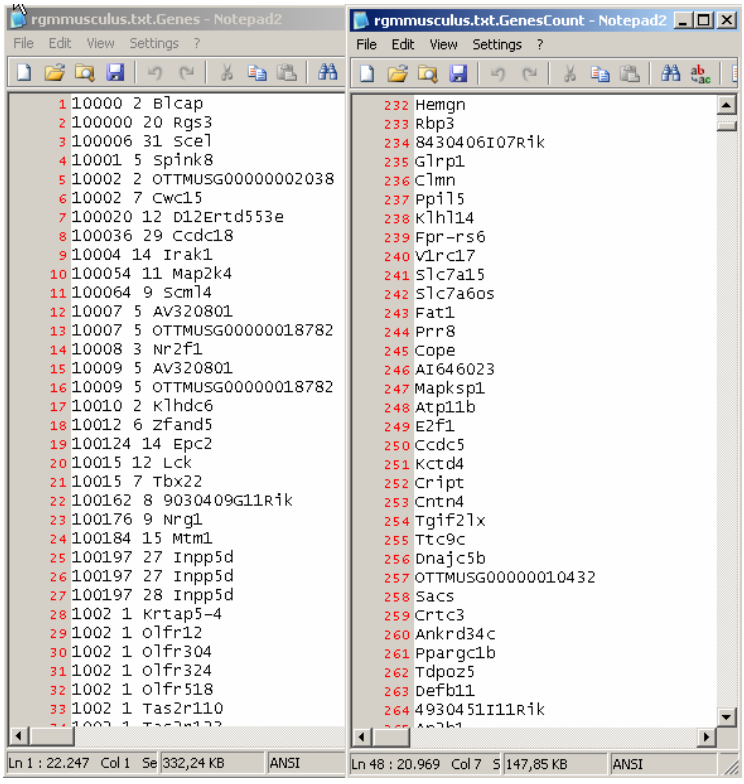
```
#!/bin/bash

#************************************************
#borrando los ficheros de pruebas anteriores
rm rg*S1*
rm rg*Genes*
rm rg*GenesCount*



for fn in `ls rg*`
  do

  #Se repite para cada fichero rg* en el diretorio del
  #script. en la variable fn se guarda el nombre del
  # fichero en curso.................................

  echo "*************fichero  $fn            **********"
  echo "*************CONTEO DE EXONES Y TAMAÑO**********"
  #contar exones y su promedio

  #l es el contador de exones >>>en el campo $9
  #m es el contador de longitud >>> campo $6-$5
  #NR es una variable interna que nos da el numero de filas
  gawk 'BEGIN{l=0;m=0} {l=l+$9;m=m+($6-$5)}
                 END {print FILENAME, "número Exones =", l,
                 "promedio Exones =",l/NR,
                 "promedio Longitud =", m/NR}' $fn


  #echo "*************ORDENANDO LOS FICHEROS*************"
  #wc -l $fn  ##me sirve para comprobar descomentando
  #echo "************************************************"

  FichOUT=$fn.S1
  sort -n $fn > $FichOUT
  fichIN=$fn.S1
  FichOUT=$fn.Genes


  #lOS FICHEROS *.Genes me sirven para comprobar que estoy
  #haciendo las cuentas bien.............................

  #
  gawk '{print $6-$5,$9,$13}'  $fichIN   | sort  > $FichOUT

  #echo "************************************************"
  fichIN=$fn.Genes
  FichOUT=$fn.GenesCount

  #los fichero *.GenesCount continen una linea por gen con
  # Su nombre correspondiente.
  # uniq fichero : quita los repetidos

  gawk '{print $3}' $fichIN | uniq   > $FichOUT
  echo "*************FIN ORDENANDO LOS FICHEROS*********"
  echo
done

# Presenta en pantalla el conteo de genes

echo "************* NUMERO DE GENES *********"
wc rg*Genes*
```

Ln 9 : 63 Col 1 Sel 0 | 1,72 KB | ANSI | LF | INS | Default Text

Y en la pantalla:



Y una muestra de los ficheros intermedios.

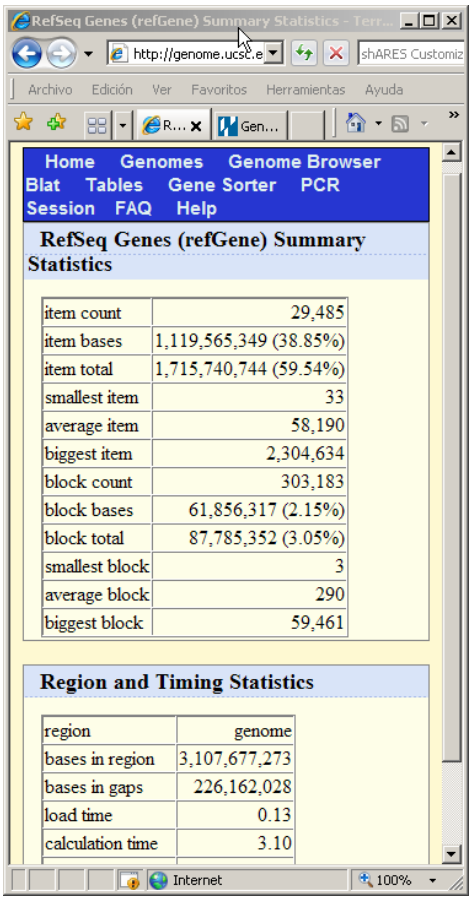


>> Estos ficheros los uso para comprobar que la salida que obtengo es realmente la que busco con los comandos.

5. Resultados finales

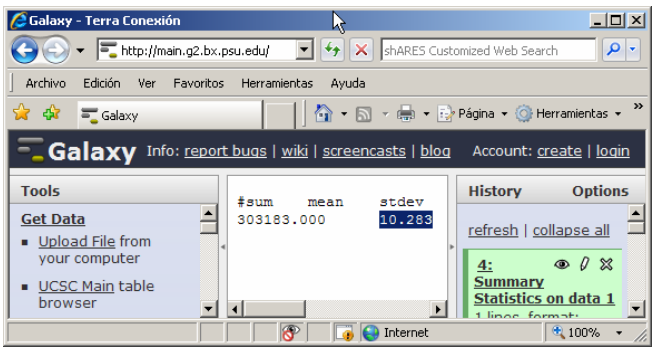
	Nº de transcritos	Promedio longitud transcritos	Promedio Exones por transcrito	Nº de genes
H. sapiens	29363	58159	10,298	23886
M. musculus	22246	47156	9,847	20968
G. gallus	4355	27897	9,871	4329
D. melanogaster	25591	11140	4,338	19401
C. elegans	24987	3300	6,395	23787

(Algunos valores los he redondeado)



6. Interpretación biológica de los resultados.

- Como demuestran los datos, el número de genes no es proporcional a lo desarrollado de una especie. Como curiosidad el C. elegans (un pequeño gusanito) tiene casi tantos genes como los humanos y mas de 4 veces que los gallos.
- El organismo comparado con las estadísticas (se entiende que son las de la tabla de la sección 5) más similares a los humanos es el ratón (m. musculus). De ahí que sea uno de los organismos mayormente utilizados en laboratorio para experimentación.
- Parece que el mejor parámetro para medir la complejidad bioquímica de un organismo es el promedio de exones por trascrito. Hay claramente dos ordenes de magnitud: El de los vertebrados (sobre 10) y el de los insectos (entre 6 y 4).
- Igualmente, un organismo complejo muestra una mayor longitud de los transcritos. Este hecho se constata no solo por los resultados obtenidos sino también por la cantidad de citas bibliográficas que emplean este parámetro.
- El número de trascritos en principio no puede ser comparable, ya que puede depender de la cantidad de análisis que se haya hecho del organismo, y del estado de resolución experimental de su genoma.

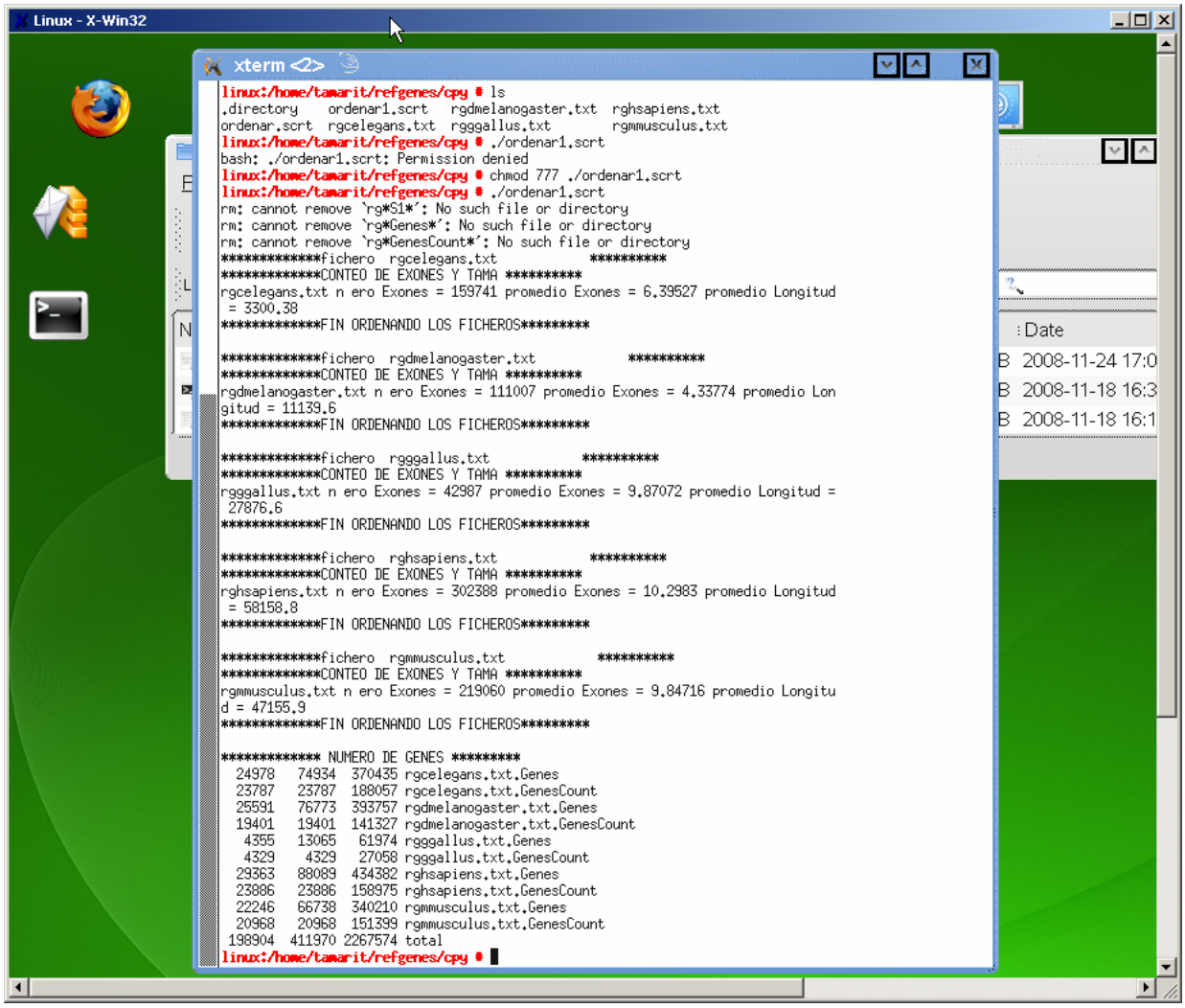


Comprobación a 28.11.2008 de RefGene para H. sapiens. Vemos que ya ha crecido el numero de registros en de 29363 a 29485, el promedio de longitud es 58190 (58159 en nuestro cálculo). Y la media de exones por trascrito es de 10.28 (desde galaxy) y coincide con nuestros cálculos.

Probando todo en el OpenSuse Linux.

Para comprobar que todo el trabajo es reproducible lo ejecuto desde el OpenSuse que tengo instalado. Muevo con el winspc al home y allí ejecuto el ordenar1.scrt

Y vemos que da lo mismo.



Ejercicio 2 – Anotación funcional de genes

Ejercicio 2 – Anotación funcional de genes [30%]

GeneOntology es un diccionario de conceptos biológicos que son utilizados para definir las funciones de los genes. Echale un vistazo a estas páginas y describe en pocas líneas en qué consiste:

- http://www.geneontology.org/
- http://en.wikipedia.org/wiki/Protein_ontology

Ahora vamos a trabajar con la anotación funcional GO del genoma de la mosca de la fruta. En este caso, en lugar de usar los identificadores NM, utilizaremos el fichero de genes de la base de datos de esta especie (FlyBase). Adjuntos a este enunciado teneis los ficheros (comprimidos) `gene_ontology.dat` y `gene_association.fb` (ambos de texto sin formato).

El fichero `gene_ontology.dat` contiene todas las definiciones (indexadas por su GOid) que pueden asignarse a los genes:

GO:0000001 mitochondrion inheritance
GO:0000002 mitochondrial genome maintenance
GO:0000003 reproduction

El fichero `gene_association` contiene las definiciones GO que están asociadas a los genes de la mosca de la fruta (*D.melanogaster*):

```
FB       FBgn0000139      ash2              GO:0048096 FB:FBrf0085406|PMID:8555105 IMP P
absent,    small,    or    homeotic    discs    2    1124/11|291.8|703|ASH2|CG6677|ash-
2|l(3)11241|l(3)S11241|l(3)SG65|mad|many  abnormal  discs     gene          taxon:7227
20060803 FlyBase
```

Debeis contestar las siguientes preguntas usando los comandos apropiados de LINUX en vuestro terminal:

- ¿ Cuántos y cuáles genes de la mosca están relacionados con la regulación de la transcripción?
- ¿ Cuántos y cuáles genes están asociados a los discos imaginales de ala de la mosca?

NOTA: Podeis ver qué es un disco imaginal en la siguiente dirección:
http://es.wikipedia.org/wiki/Disco_imaginal

Definición de GeneOntology

El proyecto GO es básicamente un vocabulario de términos biológicos que describen los genes y los productos génicos, y su regulación, estableciendo anotaciones de los términos a la información disponible.

El vocabulario consta de tres diccionarios de términos u ontologías:

- Función molecular de los genes y sus productos (proteinas, mARN, etc): Ej: “Actividad catalítica”,
- Procesos biológicos Ej: “Transporte alfa-glucosidico),
- Componentes celulares. Ej”Ribosoma”,

Cada entrada del diccionario tiene asociado un índice (GOid) único sus sinónimos y su definición. Cada uno de los términos está asociado con el resto como un grafo acíclico, de forma que se pueden establecer relaciones diversas entre los términos (un grafo acíclico no tiene principio ni fin y a un vértice se puede llegar desde distintos caminos).

Por otra parte, se desarrollan bases de datos de estudios genéticos de los organismos en donde se incluyen las asociaciones de los productos génicos con los términos go. Una entrada en la tabla de asociaciones: En donde se indica principalmente (en el siguiente apartado incluyo los campos de la tabla):

- El gen o producto génico relacionado,
- El GOid,
- Quien y cuando ha hecho la anotación,
- La base de datos donde localizar la información,
- La evidencia de la anotación.

Mediante el uso combinado de las tablas de términos y de anotaciones podemos realizar consultas que nos ayudarán a localizar toda la información relacionada con un termino Go.

El formato de anotación GO

De la web de UCSC busco la definición de la tabla que usaremos más adelante. Los campos campos

```
The gene_association.cgd.gz file uses the standard file format for
gene_association files of the Gene Ontology (GO) Consortium.  A
more
complete description of the file format is found here:

http://www.geneontology.org/doc/GO.annotation.html#file

Columns are:                                        Contents:

 1) DB                                              - database contributing the
file (always "CGD" for this file)
 2) DB_Object_ID                                - CGDID
 3) DB_Object_Symbol                                - see below
 4) Qualifier                          (optional)       - 'NOT', 'contributes_to', or
'colocalizes_with' qualifier for a GO annotation, when needed
 5) GO ID                                           - unique numeric identifier
for the GO term
 6) DB:Reference(|DB:Reference)                     - the reference associated
with the GO annotation
 7) Evidence                                        - the evidence code for the
GO annotation
 8) With (or) From                  (optional)      - any With or From qualifier
for the GO annotation
 9) Aspect                                          - which ontology the GO term
belongs in (see note beow)
10) DB_Object_Name(|Name)  (optional)          - a name for the gene product in
words, e.g. 'acid phosphatase'
11) DB_Object_Synonym(|Synonym) (optional)     - see below
12) DB_Object_Type                                 - type of object annotated,
e.g. gene, protein, etc.
13) taxon(|taxon)                              - taxonomic identifier of species
encoding gene product
14) Date                                           - date GO annotation was made
15) Assigned_by                                    - source of the annotation
(always "CGD" for this file)

Note on CGD nomenclature (pertaining to columns 3 and 11):

Column 3 - When a Standard Gene Name (e.g. CDC28, COX2) has been
        conferred, it will be present in Column 3. When no Gene Name
        has been conferred, the ORF Name (e.g., orf19.6632) will be present in
column 3.

Column 11 - The ORF Name (e.g., orf19.6632) will be the first name present
        in Column 11. Any other names (except the Standard Name, which will
        be in Column 3 if one exists), including Aliases used for the gene
        will also be present in this column.

Note on Aspect (column 9):
     C = Cellular Component
     F = Molecular Function
     P = Biological Process
```

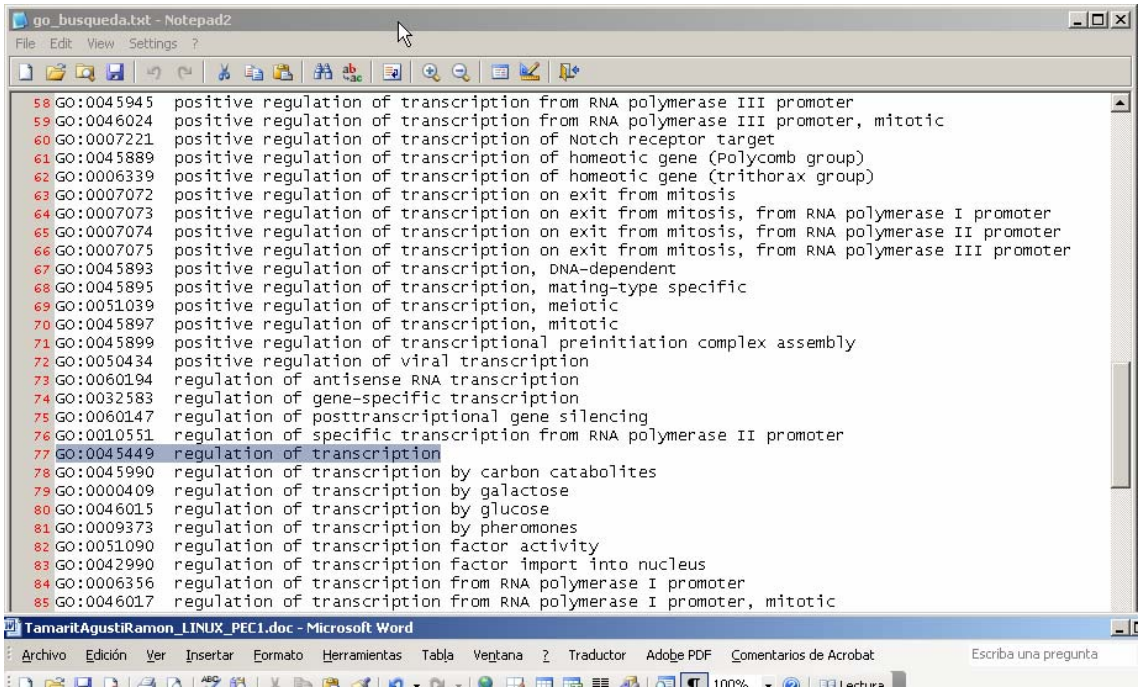
7. Genes de la mosca implicados en la regulación de la trascripción.

Primero buscamos la palabra “transcription” y “regulation” en el diccionario

```
tamarit@of_extrusion1 ~/go
$ grep transcription gene_ontology.dat>go_trasncription.txt
```

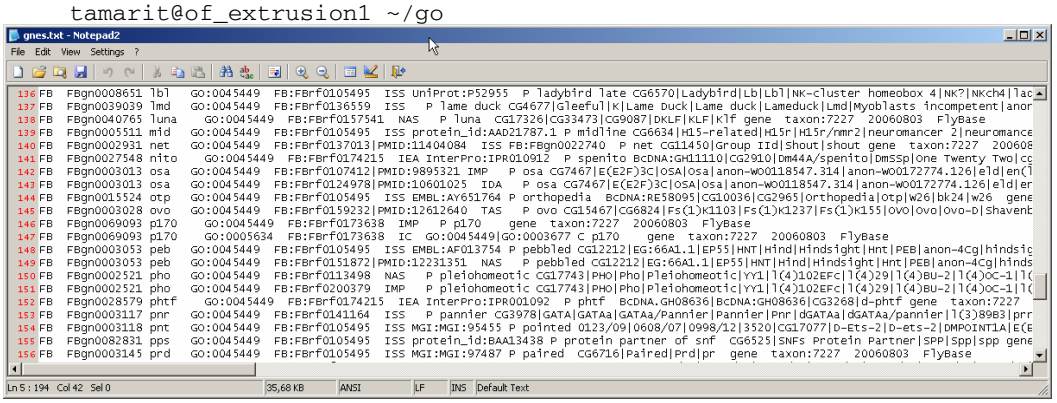
(grep nos separa las líneas que contienen el argumento “gene” y lo guardamos en un fichero de texto desviando la salida con “>”)

```
tamarit@of_extrusion1 ~/go
$ grep transcription gene_ontology.dat| grep regulation>go_busqueda.txt

tamarit@of_extrusion1 ~/go
$
```



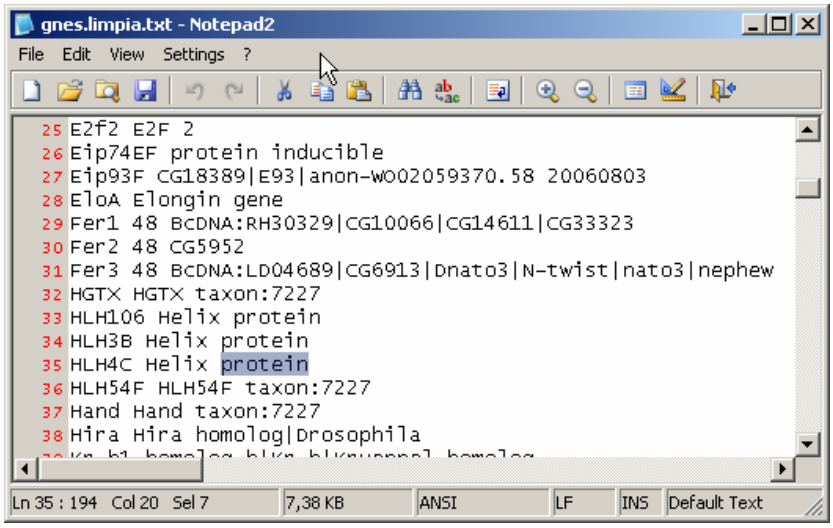
La salida anterior nos sirve para identificar el GOid que usaremos en la búsqueda de anotaciones.

```
Separamos las entradas con GO:0045449
tamarit@of_extrusion1 ~/go
$ grep GO:0045449 gene_association.fb>gnes.txt

tamarit@of_extrusion1 ~/go
```



$

Limpiamos un poco la tabla para verla mejor

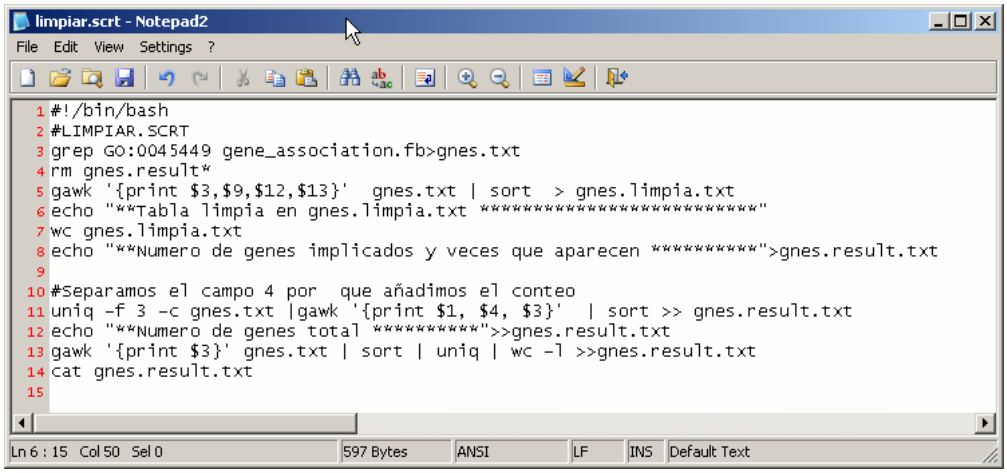


¡¡Ojo!! por que no todos son genes hay de todo (veasé la figura de arriba “protein”). Si nos fijamos en el formato de gene_association.gb, vemos que los genes están identificados con la palabra “gene” en la columna 12:

```
10) DB_Object_Name(|Name)  (optional)          - a name for the gene product in
words, e.g. 'acid phosphatase'
11) DB_Object_Synonym(|Synonym) (optional)     - see below
12) DB_Object_Type                                    - type of object annotated,
e.g. gene, protein, etc.
13) taxon(|taxon)                              - taxonomic identifier of species
encoding gene product
```

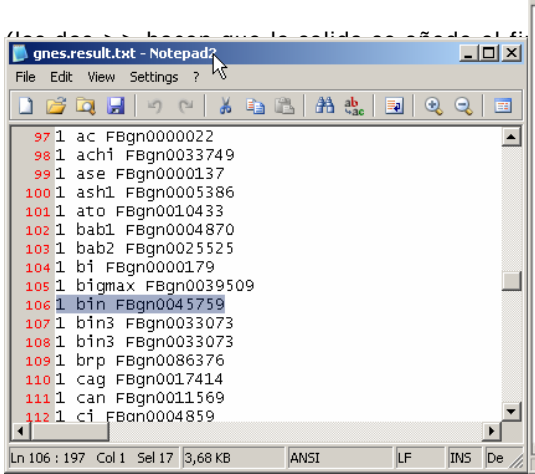
Procedemos a separar los “gene”:

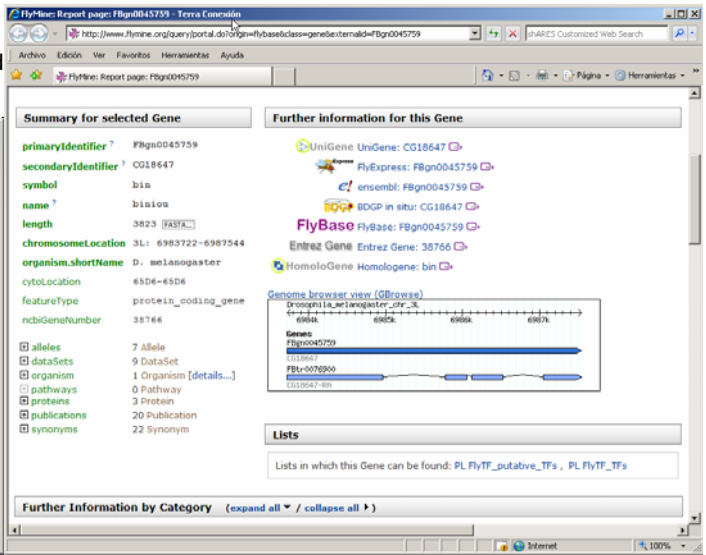
Preparo un script limpiar.scrt . Al final el script queda



(gawk ‘{print $i….}´ imprime los campos seleccionados del fichero) (sort los ordena) (wc cuenta las líneas, palabras y caracteres)

(uniq –f 3 …. Hace que se eliminen los repetid



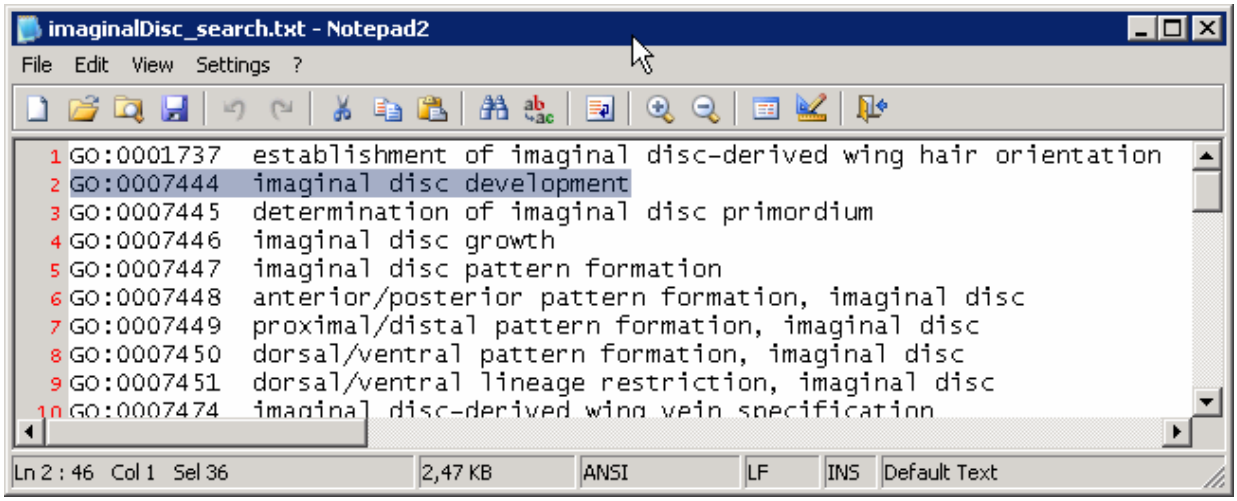


Genes asociados a los discos imaginales

Repetimos el proceso anterior pero con las palabras “imaginal discs”

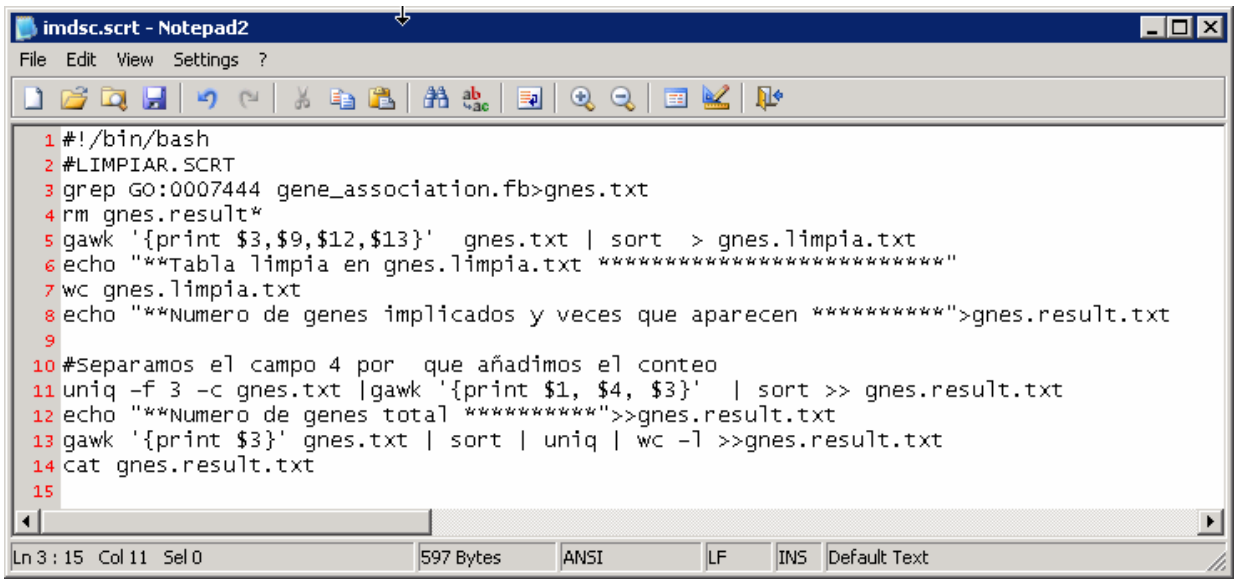
```
tamarit@of_extrusion1 ~/go

$ grep imaginal gene_ontology.dat| grep disc > imaginalDisc_search.txt

tamarit@of_extrusion1 ~/go
```

Visualizamos el resultado y comprobamos que existe una entrada igual a la que buscamos:



La entrada que nos interesa lleva el Goid GO:0007444.

Retocamos el script del punto anterior. Lo renombramos a: imsrch.scrt



(Con el comando cat muestro el resultado contenido en el fichero gnes.result.txt)

(Cuando utilizo “>>” es para que no me genere un uno archivo, sino que ponga la salida al final)

El resultado:

En total hay 31

<<<< el conteo de genes se hace con gawk'{print $3}´ gnes.txt |sort | uniq | wc –l (en realizad el sort sobra, pero como es rápido tampoco me pare a bórralo. El script se podría optimizar para que fuera más rápido si lo quisiéramos volver a usar). La utilidad de hacerlo así es que no hay que estar tecleando series de comandos continuamente para hacer comprobaciones, lo podemos reutilizar, o incluso añadirle modificaciones más adelante.

Comprobaciones de los cálculos

Vamos a web de amigo y filtramos por imaginal disc development en drosophyla y que solo nos muestre los genes:

Guardamos el fichero y comprobamos:

Obtenemos 36 entradas 5 mas. Pero en realidad si comprobamos el campo:

```
6) DB:Reference(|DB:Reference)
- the reference associated with the GO
annotation
```

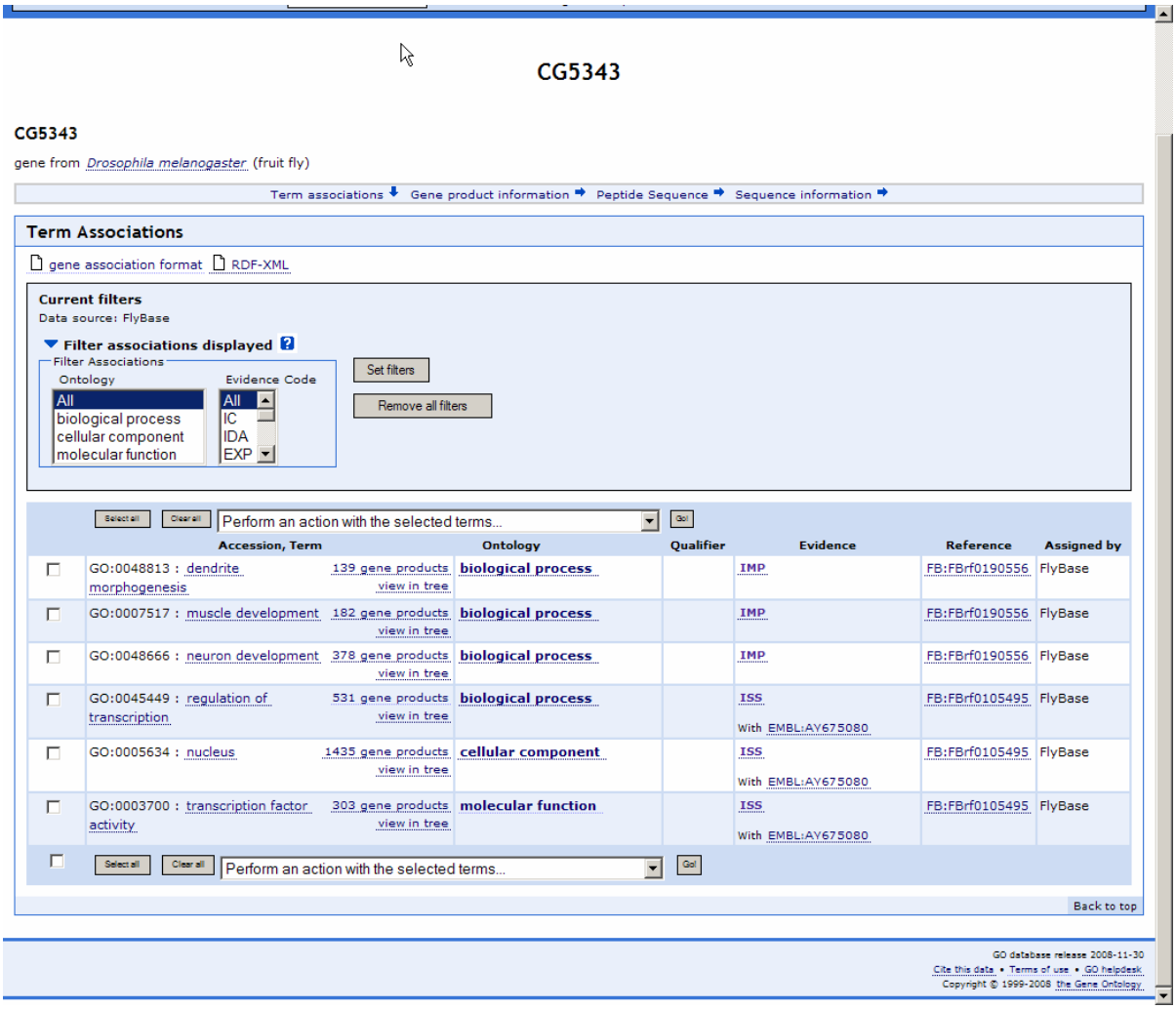
Vemos que en realidad son 31 entradas. Están repetidas por que existen varias referencias (experimentaciones) asociadas con el mismo elemento y además con “evidencias diferentes”.

En el caso de “Trasncription Regulation”, seguimos el mismo proceso, aquí la base nos devuelve 149 registros, menos de los que nos da a nosotros. Si cojemos uno de ellos:

FB FBgn0032248 CG5343 GO:0045449 FB:FBrf0105495|EMBL:AY675080 ISS EMBL:AY675080.

Que si nos sale a nosotros pero no en el listado que nos hemos bajado del amiGO. Vamos a la pagina de amiGO y lo comprobamos. Nos llevamos la sorpresa , encontramos su entrada, pero no asociada a al GO:0045449, sino a otros más específicos.

Pero si buscamos específicamente su entrada: Si que lo encontramos y ademas relacionado a “regulation of trasncription”. Nos quedamos tranquilos de que lo que hemos hecho esta bien.

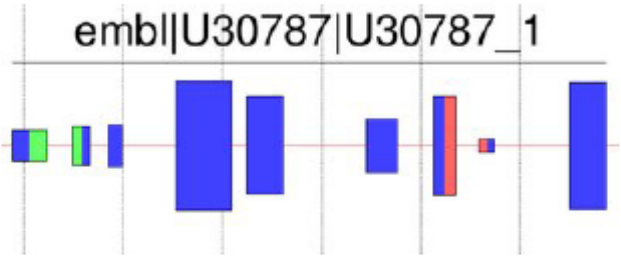


Ejercicio 3 – Predicción computacional de genes

Ejercicio 3 – Predicción computacional de genes [30%]

(A) Un investigador nos comenta que la secuencia `U30787.fa` (adjunta a este enunciado) contiene un gen humano. Debeis usar el programa **geneid** en vuestro terminal para averiguar cuál es la estructura éxonica más probable de ese gen según este programa.

(B) Ahora debeis representar gráficamente este gen usando el programa **gff2ps**, explicadme como obteneis una imagen como ésta y qué significan los colores asignados a los exones:



(C) Finalmente, antes de mostrarle nuestra predicción a nuestro colaborador, nos gustaría convertir esta imagen a un formato más atractivo. Cread un fichero de parámetros para **gff2ps** que reproduzca un formato como éste:

ATENCIÓN: Como Título de la imagen (en lugar de Genefinding on your computer) escribid vuestro nombre y el de la asignatura.

MATERIALES:

Recordad que las predicciones (el gen detectado por geneid) debe estar en formato GFF. Aquí teneis la descripción de éste, columna por columna:
http://genome.imim.es/software/geneid/docs/chapter4/formats_html/gff.html

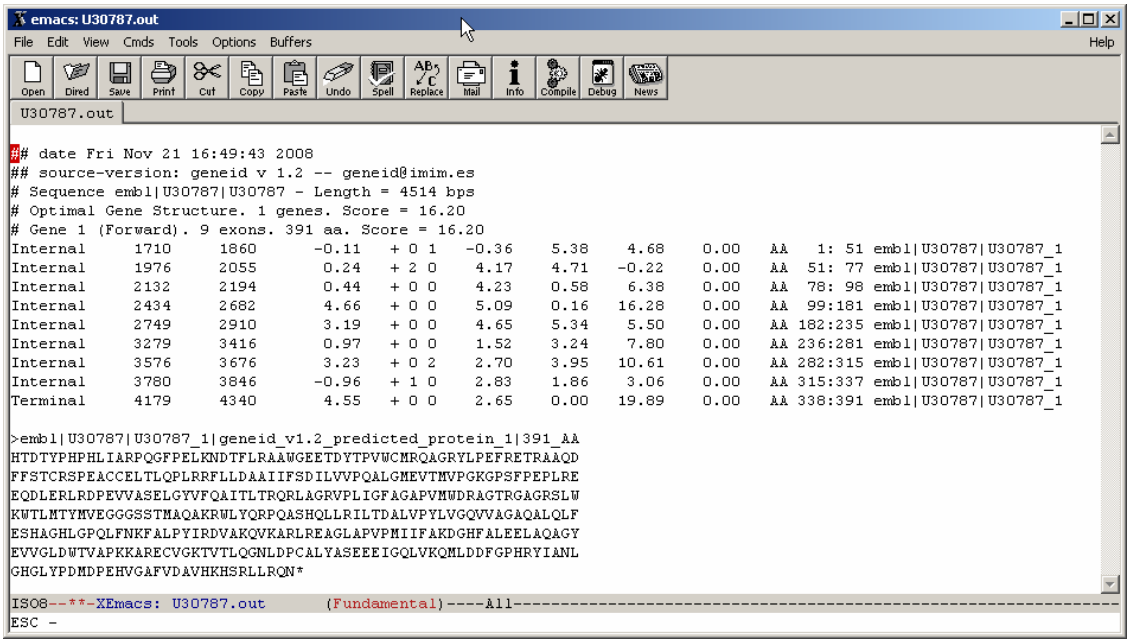
Para llamar a GFF2PS con un fichero de parámetros "gff2ps.param", debeis hacer:
% gff2ps –C gff2ps.param predicciones.gff

Os adjunto al enunciado un fichero de parametros semivacio para que veais el formato. Asimismo teneis tambien el manual de usuario del programa GFF2PS (paginas 16-26 o apéndice B para las definiciones de campos).

Nota.- He usado el geneid sobre cygwin- Para hacerlo funcionar hay que compilarlo de una forma un poco especial ya que el makefile por defecto da errores de linkado. Adjunto como anexo las indicaciones para compilarlo (como geneid.exe) para cygwin.

a) Estructura exónica más probable.

```
tamarit@of_extrusion1 ~/geneid

$ ./geneid -P param/human3iso.param U30787.fa > U30787.out
```



La interpretación del resultado es la siguiente:

- Se predice 1 gen con 9 exones en dirección “forward” (5’->3’), con un score de 16.20 (que es la suma de los scores de de los 9 exones)
- Todos los exones son de tipo “internal” o internos

  Se pueden encontrar cuatro tipos (los nuestros son internal):

  - Iniciales: Delimitados por un codón inicial (ATG) y un donnor site.
  - Internos: Delimitados por un “acceptor site” y un” donnor”.
  - Terminales: Delimitados por un donnor site y un codón STOP (TGA, TAG y TAA).
  - Singles: Delimitados por un codón inicial y un STOP.
- Las coordenadas internas de cada exón están en las columnas 2 y 3
- La probabilidad (store) de cada exón se obtiene de la columna 4.
- El + indica que todos van en dirección “forward”
- El frame del exón esta el campo 6, el remanente “remaninder” en el 7,
- Los campos siguientes 8,9,10,11 son scores variados (ver en el manual).
- La localización de cada exón en la secuencia de aminoácidos de la predicción esta en los campos 12 y 13 (inicio y fin)
- El ultimo campo es el identificador del gen.
- Debajo tenemos la proteína predecida en formato fasta.

Con el parámetro –X se puede obtener una salida más completa.

b) Representación gráfica del gen

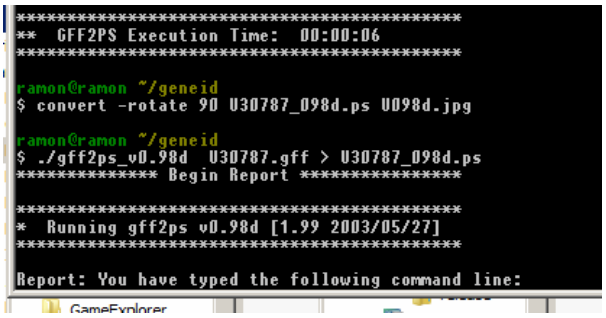
Preparamos la salida para el gff2ps

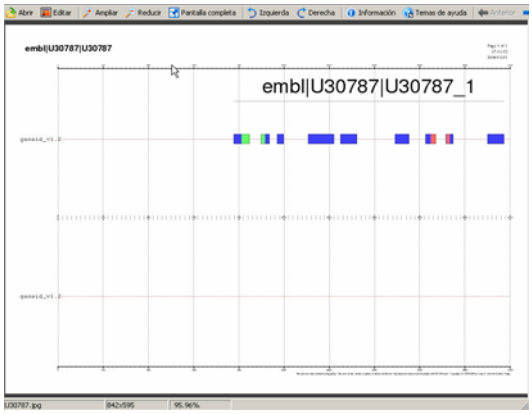
tamarit@of_extrusion1 ~/geneid

$ ./geneid -vGP param/human3iso.param U30787.fa > U30787.gff

Y lo convertimos a jpg con

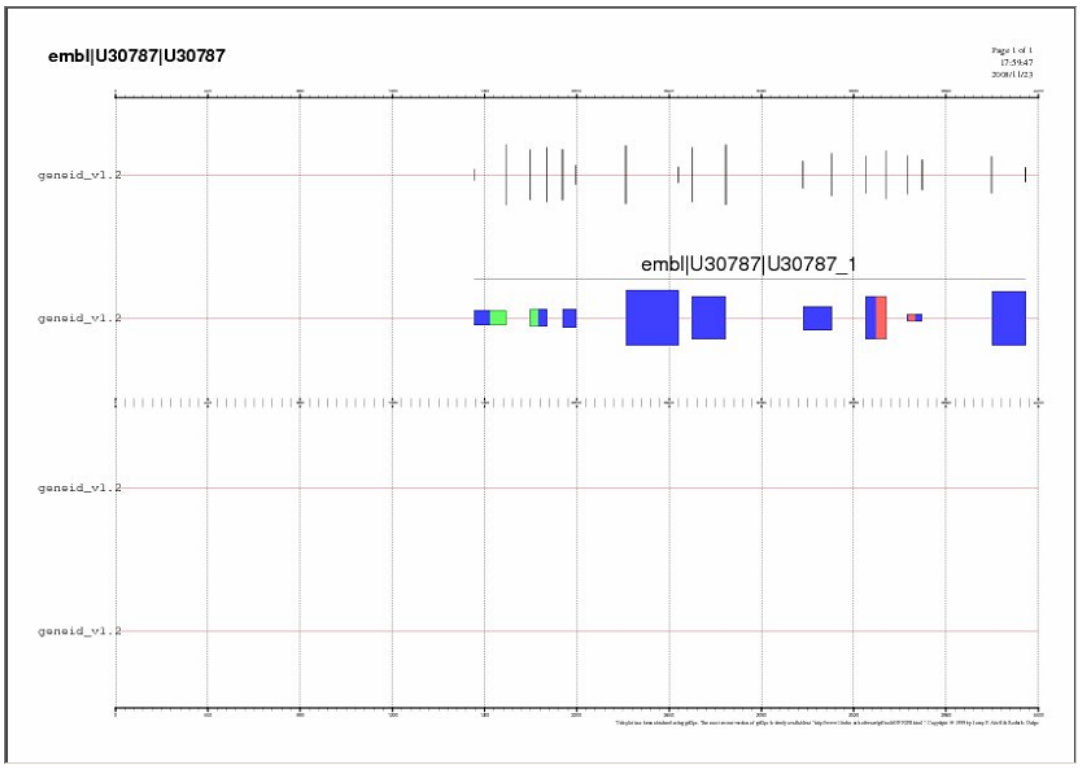
```
tamarit@of_extrusion1 ~/geneid

$ convert -rotate 90 U30787.ps U30787.jpg
```





Esta versión parece que no saca los scores correctamente, así que descargo la versión 098d y pruebo.

Y el resultado: Esta es la imagen que buscábamos. (abajo)



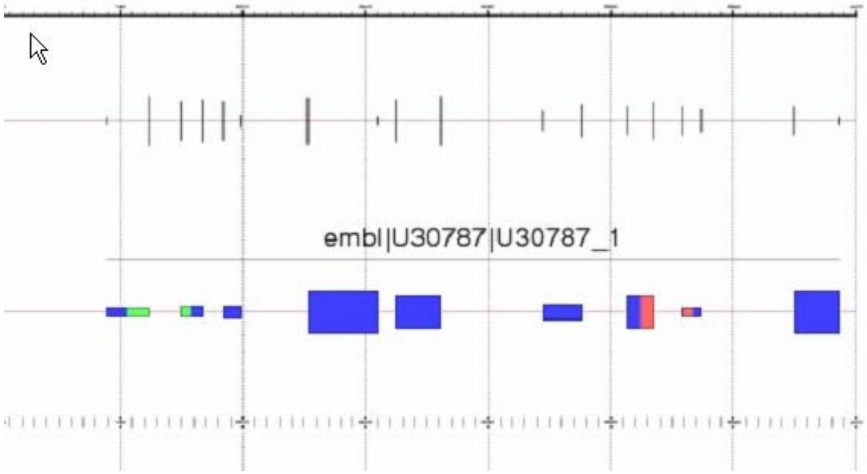
En el siguiente ejercicio comprobamos que podemos obtener la misma imagen modificando la salida con un fichero de parámetros modificando los valores por defecto de:

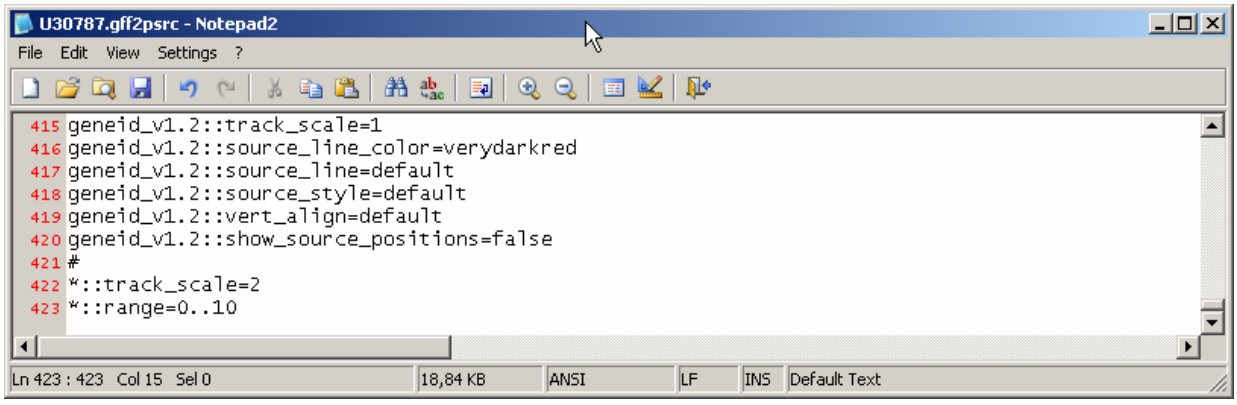
Por ejemplo

*::track_scale=2
*::range=0..20
Comprobado

De esta forma podemos usar la versión 098l.>>>

Imagen de la derecha obtenida con el fichero de parámetros modificado.





Significado de los colores

El significado de los colores por defecto esta especificado en el manual, e indica el frame del exon y del "remainder" (Exon a la izquierda – remainder a la derecha). Sirve para comprobar visualmente la consistencia de los frames de exones adyacentes. Dos exones son compatibles si los colores de sus caras enfrentadas (las cajas) son del mismo color (tiene el mismo frame).

Los colores que se definen por defecto son:

frame0_color=blue (Azul)

frame1_color=red (Rojo)

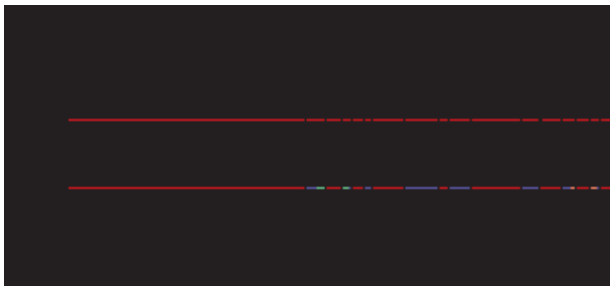
frame2_color=green (verde)

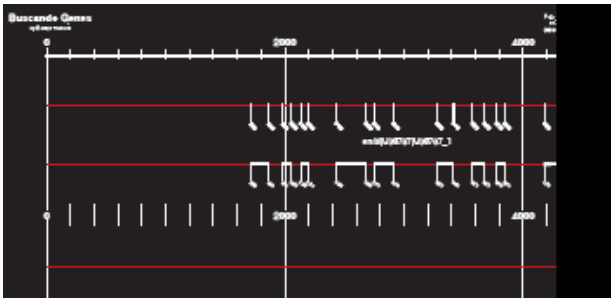
Para calcular el frame de los posibles exones a partir de las señales de splicing hay que tener en cuenta que los codones de los exones tienen que estar en la misma pauta que el codón stop que delimita el ORF. Entonces:

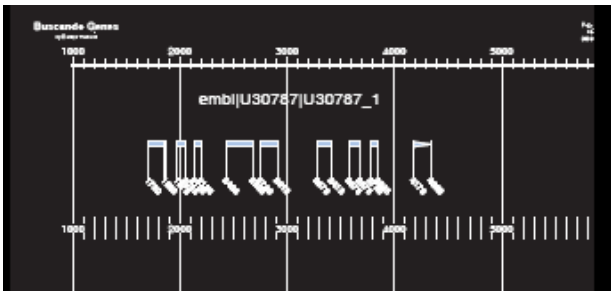
- Si el número de bases entre el inicio del ORF y el final del intrón es divisible entre 3 será frame 0.
- Si al dividir entre tres el residuo es dos: frame 1
- Si al dividir entre tres el residuo es uno: frame 2

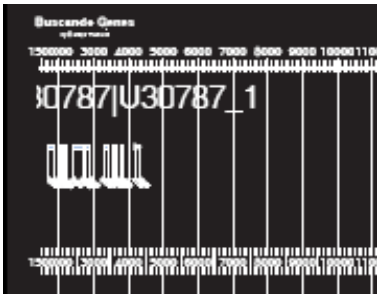
El frame de los exones iniciales y de los single (un solo exón) siempre es 0.

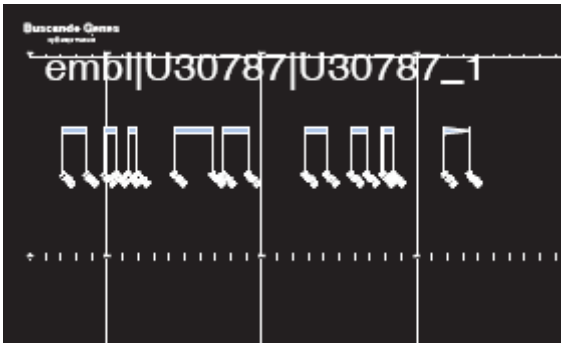
c) Imagen especial











Para resolver este problema he comenzado desde el fichero de muestra entregado con la PEC. Además me he generado un .gff2ps (fichero de parámetros por defecto) con el comando –d para ver más o menos como hay que hacer el fichero de parámetros. Igualmente me he servido del manual para ir probando las distintas opciones

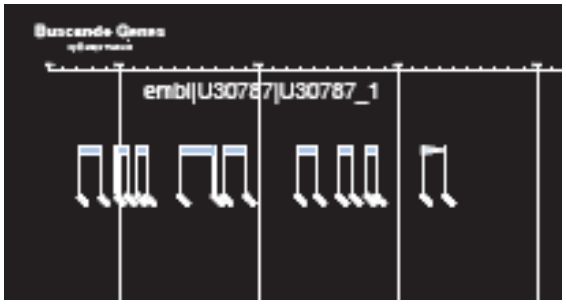
En particular me he fijado en:

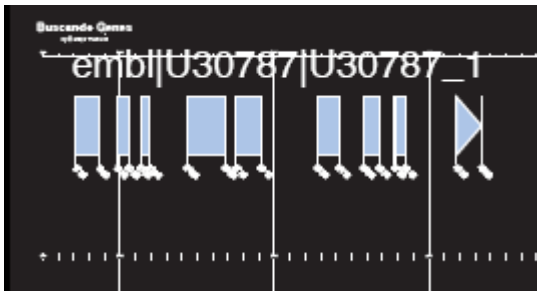
- Como cambiar el titulo
- El fondo tiene que ser negro
- Tiene que salir la escala en blanco
- Una sola secuencia en la parte superior
- Que salga las marcas de posición de los nucleotidos iniciales y finales de cada exón.
- Ajustar el tamaño de las etiquetas

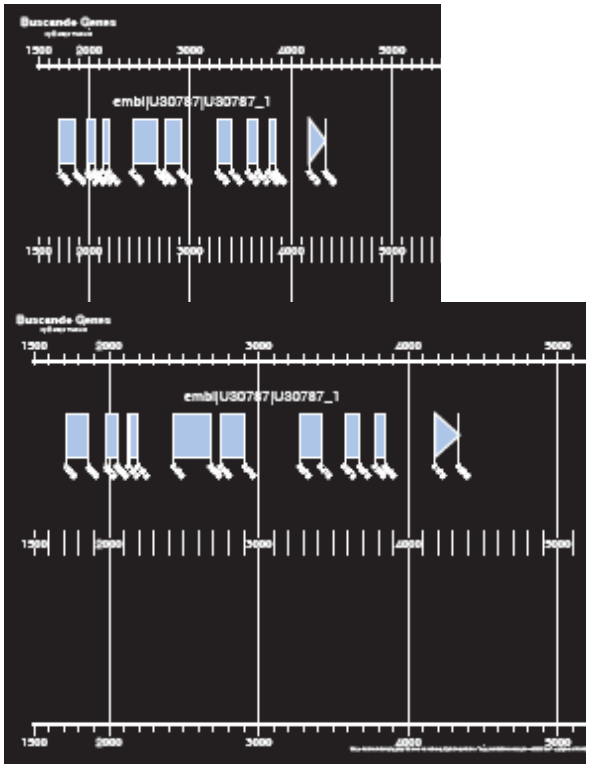
Dejo una secuencia de cómo me ha ido saliendo el dibujo a medida que cambiaba y añadía parámetros

El fichero de parámetros que al final he dado por bueno es el siguiente:

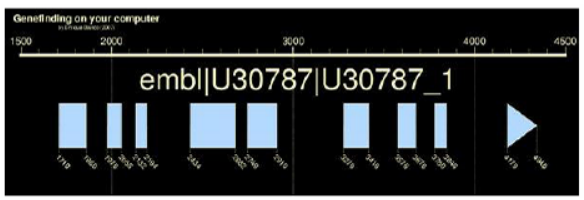
```
# L #
##pagina a4
##Para forzar a que entre todo en la
pagina
page_size=a4

## se me va hacia la derecha
##Retocamos el margen
margin_right= 2cm

##Fondo de color negro
background_color=black

##bajo fondo de lineas en blanco
##¡¡¡sino no se ven mas que puntitos
foreground_color=white

title= Ramón Tamarit - Fundamento de
informatica
subtitle=by Ramón Tamarit
##ajuste de escala
##probado con 2000 y no sale bien
major_tickmarks_nucleotides=1000
```





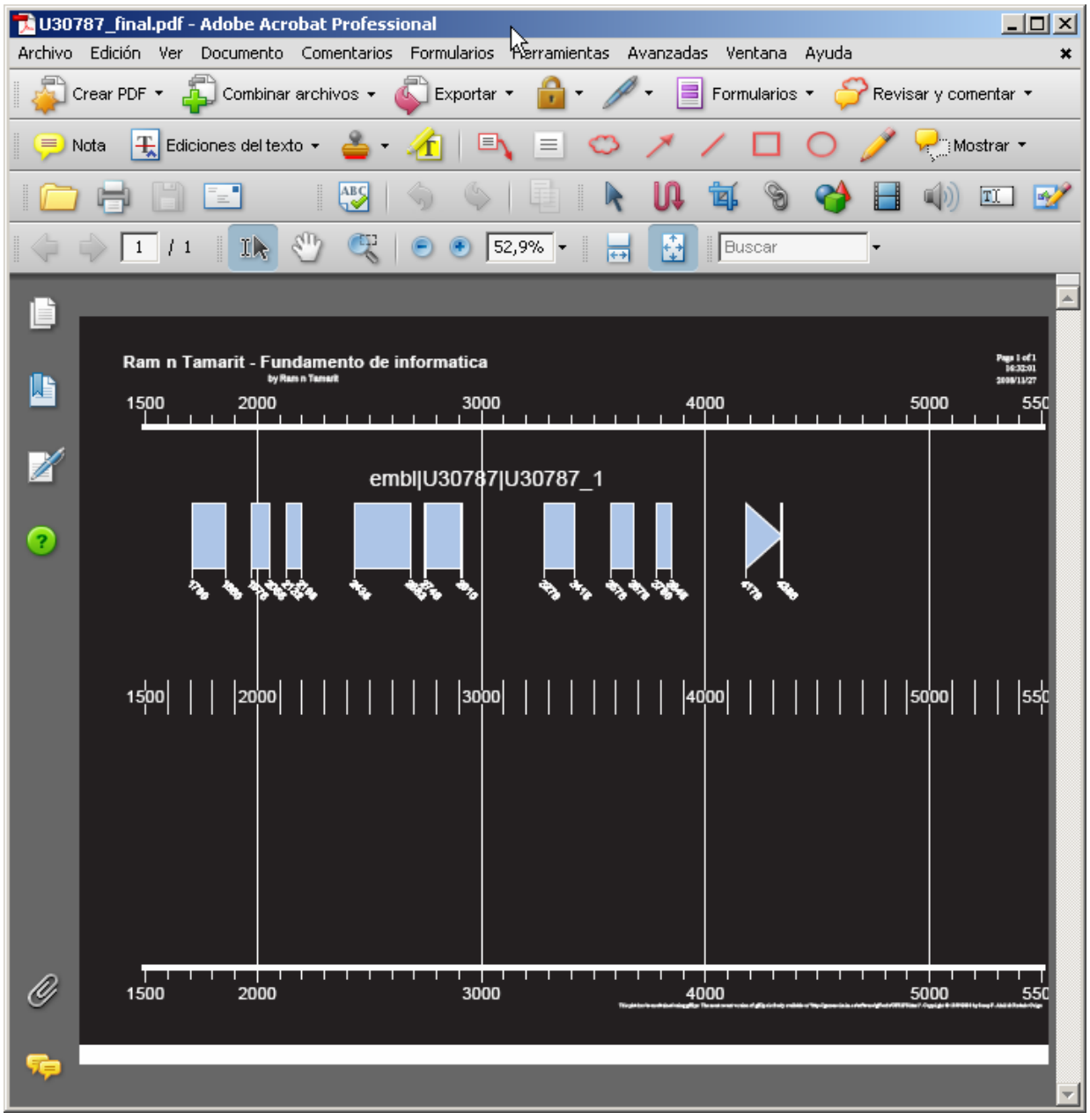


La imagen de abajo es como debe quedar la de arriba es la ultima que doy por valida a falta de cambiar el titulo.




```
##mostrar las posiciones de los exones
show_positions=on

##ponemos la escala
show_grid=on
show_outer_scale=on

## Pruebo con varios
#default_scale_width=1
#default_scale_width=2
##tamaño etiqueta escala
default_scale_width=0.8

##un solo bloque en la pagina
blocks_x_page=1

#default_track_width=0.5cm

##para acercar la secuencia a la izda
zoom=1500..5500

##ajuste de las etiquetas de escala
left_source_label_width=0.9cm
group_label_scale=1
position_label_scale=1



# G #
*::group_line=none
## todas las cajas del mismo color azulado
*::feature_color=verylightskyblue

# F #
##pone un triangulo en el exon terminal
terminal::shape=right_triangle

##los internos como caja
internal::shape=box
## que no se dividan
single::shape=single

##todos de color azul y que no muestre los traks
*::feature_color=verylightskyblue
*::fill_shape_mode=1_color

##Desctivamos los bloques reverse e
independiente
## para que no molesten
strand_show_reverse=off
strand_show_independent=off

# S #
*::left_label::++none++

##hacemos desaparecer las lineas
*::source_line=none
*::unfold_ungrouped_line=off

*::track_spacing_scale=1.5

##alargar las cajas.. y evitar que salgan los scores
*::track_scale=1
*::range=none ##antes he probado con
## 2..2 pero no salen a mi gusto
*::source_label_scale=0.3
```

Después de muchos cambio y ajustes de tamaño (en total 37 pruebas) lo dejo como en la imagen siguiente.



Ejercicio 4 – Diseño de un pipeline automático de anotación.

Ejercicio 4 – Diseño de un pipeline automático de anotación [10%]

Como ejercicio conceptual que nos ayudará a asociar este tema de LINUX con el tema de PERL, imaginaos que en lugar de hacer la anotación de un gen (como en el apartado anterior), debeis realizar este proceso para 100 genes cuyas secuencias se se encuentran en un directorio.

Debeis proponerme un script en PERL que automatice este proceso de forma que realice la predicción de los genes en esas secuencias con **geneid** y despues construya la representacion gráfica con **gff2ps**.

IMPORTANTE:
Este ejercicio numero 4 puede ser resuelto en dos modos según vuestros conocimientos prácticos adquiridos hasta ahora. Podeis animaros a implementar una versión funcional del script, o bien presentarme por escrito un esbozo de lo que creeis que debería hacer este programa (su flujo de datos). Os lo dejo a vuestra elección, el sistema de calificación en ambos casos será igual. En cualquier caso es importante que expliqueis que hace en cada instrucción este programa.

Adjunto el programa en perl. Creo que lo he comentado todo lo posible. Lo he probado y funciona.

```
#!/usr/bin/perl -w
#**********************pipegen*****************************
use strict;

# Los argumentos se ponen en la línea de comandos
# Hay que poner :
#   1.- la ruta de los archivos fasta
#       que queremos calcular,
#   2.- El archivo de parámetros de geneid
#   3.- El archivo de parámetros de gff2ps -opcional-
##por ejemplo
##pipegen ~/geneid ~/geneid/param/human3iso.param ~/geneid/.gff2ps
# Si no le llegan argumentos hay que salir del programa
# imprimiremos la variable $COMOUSAR que mostrará el un
# mensaje en pantalla
# $0 es una variable que contiene el nombre del programa

my($COMOUSAR) = "$0  - \n\t
************************************************\n\t
Para usar pipegen hay que pasar como argumentos \n\t
los ficheros a calcular \n\t
************************************************\n\t
Ejemplo:
~/geneid ~/geneid/param/human3iso.param ~/geneid/.gff2ps\n\t
************************************************\n\n";

my($LEYENDAPROCESO) = "
************************************************\n\t
PROCESANDO.....\n\n";

# @ARGV Matriz que contiene los argumentos de la linea de comandos
unless(@ARGV) {
       print $COMOUSAR;
       ## Si esta vacía salimos del programa
       exit;
          }
```

```
#inicializamos las variables
my($RutaFicheros) = $ARGV[0];
my($RutaGeneidPar) = $ARGV[1];
my $Rutagff2psPar="";
print $RutaFicheros;
chdir $RutaFicheros #nos cambiamos al directorio de trabajo
   or
die " Error - no encuentro $RutaFicheros: $!"; ##Si no podemos>> salimos

#sino damos un fichero de parametros ponemos la ruta vacia
unless(my($Rutagff2psPar) = $ARGV[2]) {
      #Ruta vacia
      $Rutagff2psPar="";
      exit;
      }


## Si le hemos puesto la ruta del archivo de parámetros de gff2ps
#   le añadimos -C
if ($Rutagff2psPar cmp "")
   {
      #le añadimos el modificador
      # el  . lo uso para unir cadenas
      $Rutagff2psPar = "-C " . $Rutagff2psPar;
      print "El fichero de parametros de gff2ps es : ", $Rutagff2psPar,"\n";

   }

##obtenemos un listado de los ficheros con extensión fasta y los guardamos
# en la matriz @Ficheros. Esta es una de las formas de llamar a la línea
# de comandos del sistema
my @Ficheros = `ls -a *.fa`;

##Inicializamos las variables escalares para hacer los inputs
#nombre del fichero fasta para el calculo
my $FicheroFasta = "";

#nombre del fichero de salida
my $FicheroFastaOUT="";
my $Ficherogff2psOUT="";

#Contador de ficheros
my $numero = 0;

#Respuesta del geneid --- log
my $FicheroLog="";


##para cada fichero de la matriz ficheros
foreach (@Ficheros) {
      #recuperamos el nombre del fichero con la variables $_
      $FicheroFasta = $_ ;

      #Aumentamos en uno el contador
      $numero ++ ;

      #eliminamos el terminador de linea \n sino luego el input de
      #geneid tendrá problemas
      chomp($FicheroFasta );

      #Preparamos el nombre del fichero de salida de geneid y gff2ps
      $FicheroFastaOUT= $FicheroFasta . ".pipegen.gff";

      $Ficherogff2psOUT=$FicheroFasta . "pipegen.ps";
```

```
###nota####
## Podriamos haber hecho las llamadas en una función para mayor
# Claridad …..

      #preparamos el input
      my $Calculo= "./bin/geneid.exe -vGP " . $RutaGeneidPar . " "
                                  . $FicheroFasta . ">"
                                  . $FicheroFastaOUT;
      print $LEYENDAPROCESO;
      print "fichero fasta  :", $numero ,"\t", $FicheroFasta,"\n";
      print "fichero param  :", $numero ,"\t", $RutaGeneidPar,"\n";
      print "comando       :", $numero ,"\t", $Calculo,"\n";
      print  $Calculo, "\n";

      #abrimos el geneid con las opciones que queremos y desviamos su
      #salida a pantalla.
      open(GENEID,"$Calculo|")|| die " Error... \n"; ## Si falla nos envía un error
      while(<GENEID>) ##mientas tengamos salida
            {
            #imprime la salida en pantalla
            #print "$_"; ##probando
            $FicheroLog = $FicheroLog . $_ ;
            }
      close(GENEID);

      #abrimos el gff2ps con las opciones que queremos

## Aquí en vez de obtener un ps para cada fichero podriamos
## haber enviado todos los *.gff a gff2ps y haber obtenido un solo ps con cada
# resultado en un bloque. Para ello habria que sacar esta parte del bucle foreach, y
## después recontruir la entrada de parámetros al gff2ps.

      $Calculo= "./gff2ps " . $Rutagff2psPar . " "
                        . $FicheroFastaOUT . ">"
                        . $Ficherogff2psOUT;
      open(GFF2PS,"$Calculo|") ||  die " Error... \n";
      while(<GFF2PS>) ##mientas tengamos salida
            {
            #imprime la salida en pantalla
            print "$_";
            $FicheroLog = $FicheroLog . $_ ;
             }
      close(GFF2PS);


      print "***************" ;
   }

##lo dejo preparado por si queremos evitar la salida
#por pantalla, guardarla en un fichero para verla después.
# en principio lo dejo comentado
#print $FicheroLog ;


exit;

############################################################
```

Compilar geneid para cygwin en windows

Descargar los programas necesarios.

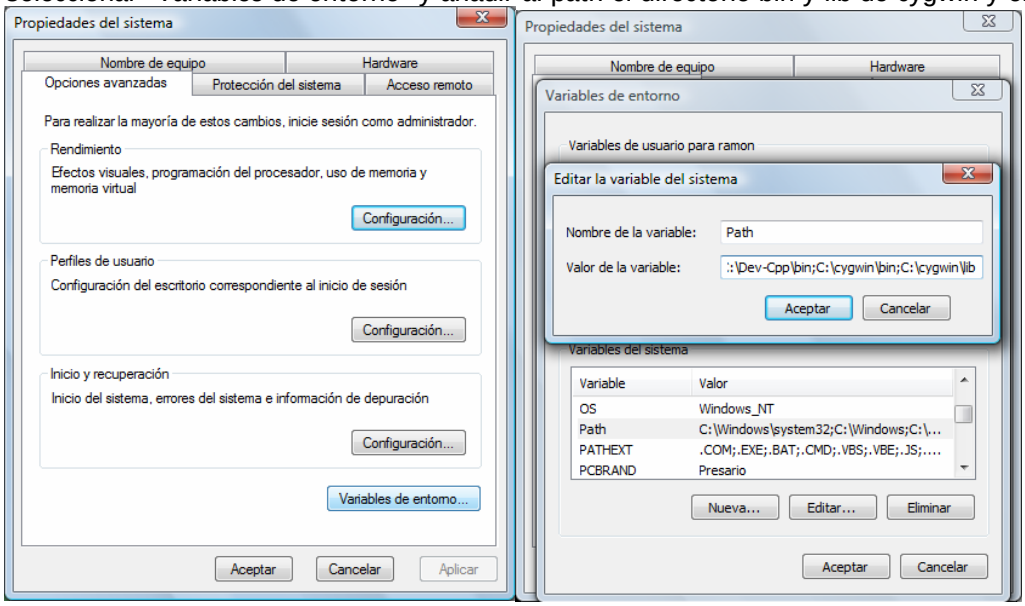
El IDE de gcc de blodshead

http://www.bloodshed.net/download.html
http://sourceforge.net/projects/dev-cpp yo he usado la versión 4.9.9.2

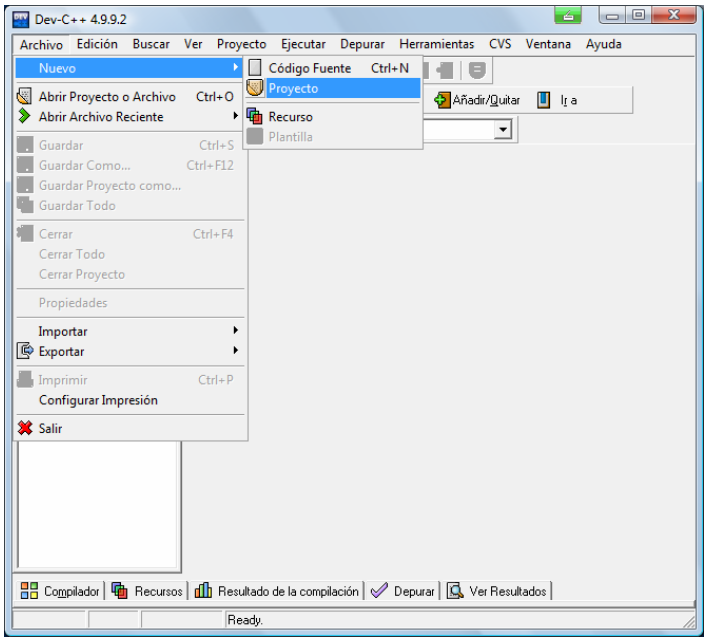
El cygwin. Asegurarse de descargar también los paquetes “devel” con el compilador.
http://www.cygwin.com/

Configurar las rutas de acceso a las libreria de sistema

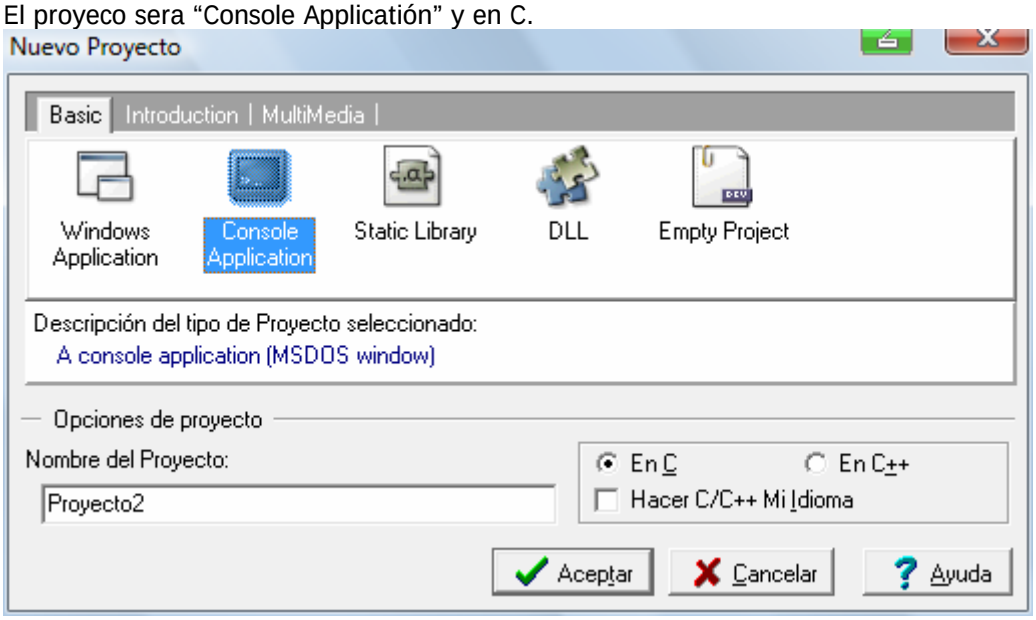
Seleccionar “Variables de entorno” y añadir al path el directorio bin y lib de cygwin y el bin del ide.



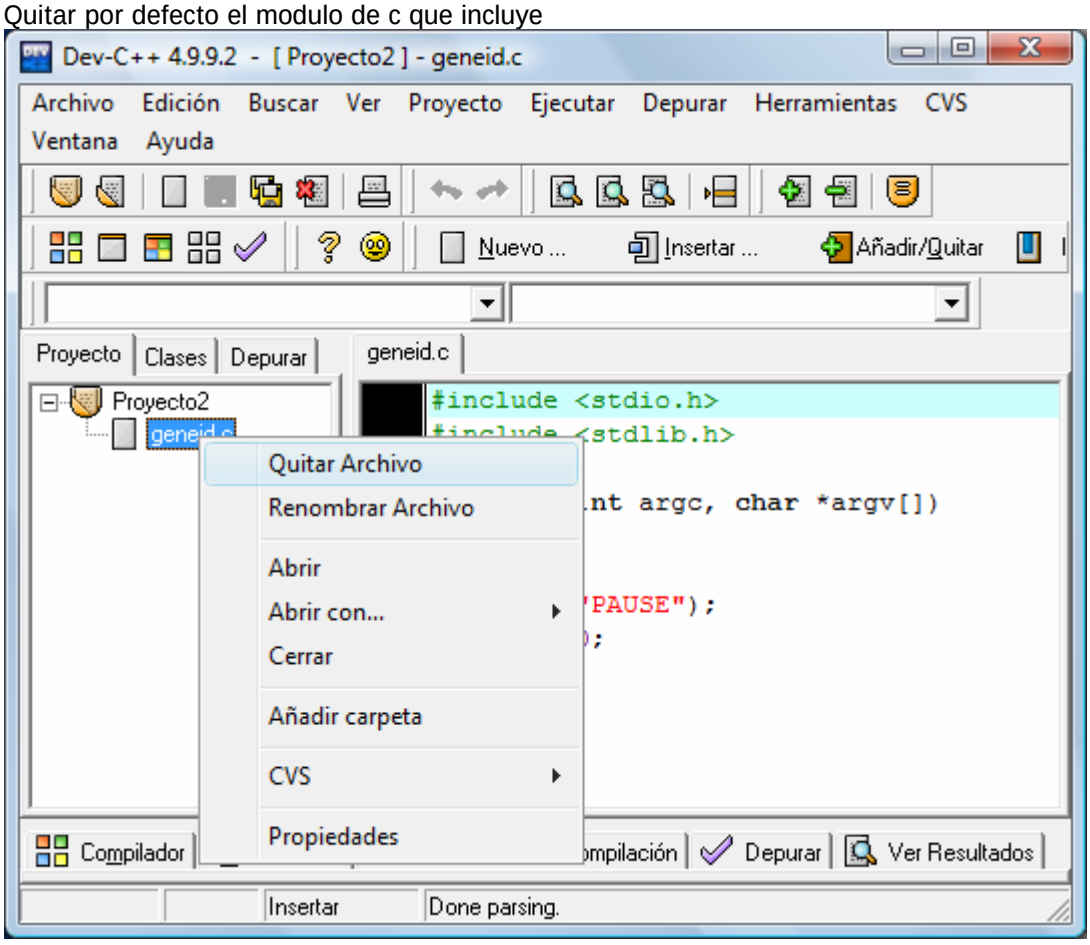
Crear un Nuevo proyecto con el Dev-C++



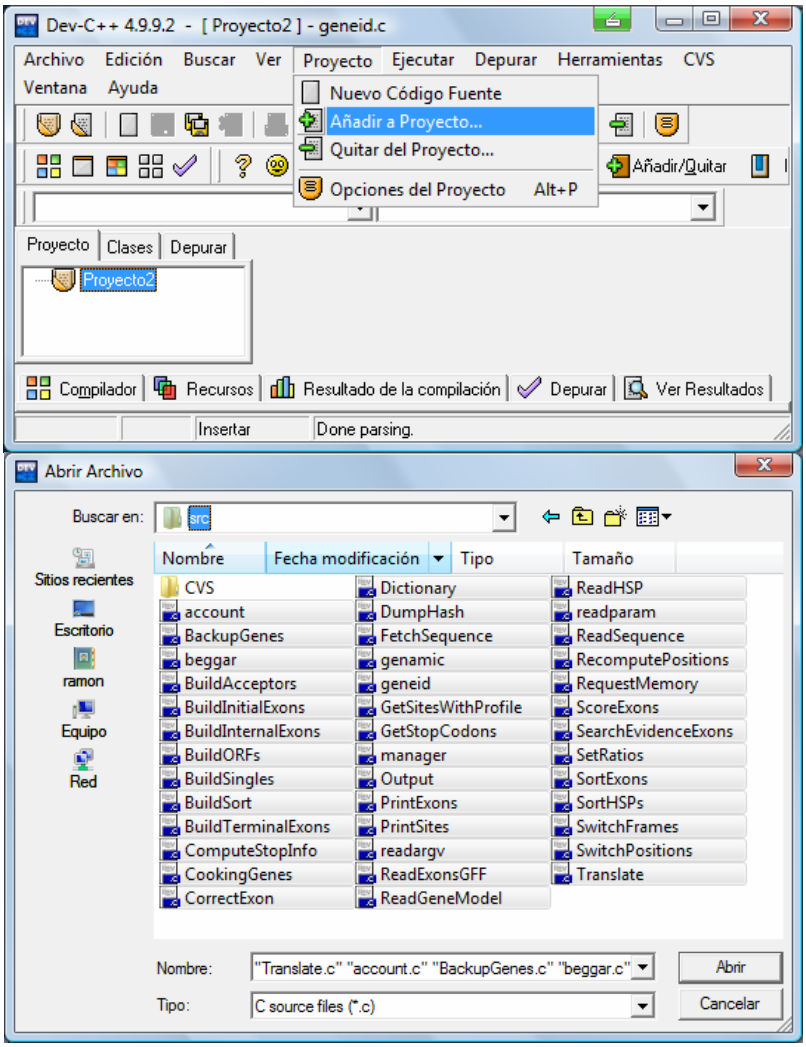
El proyeco sera “Console Applicatión” y en C.



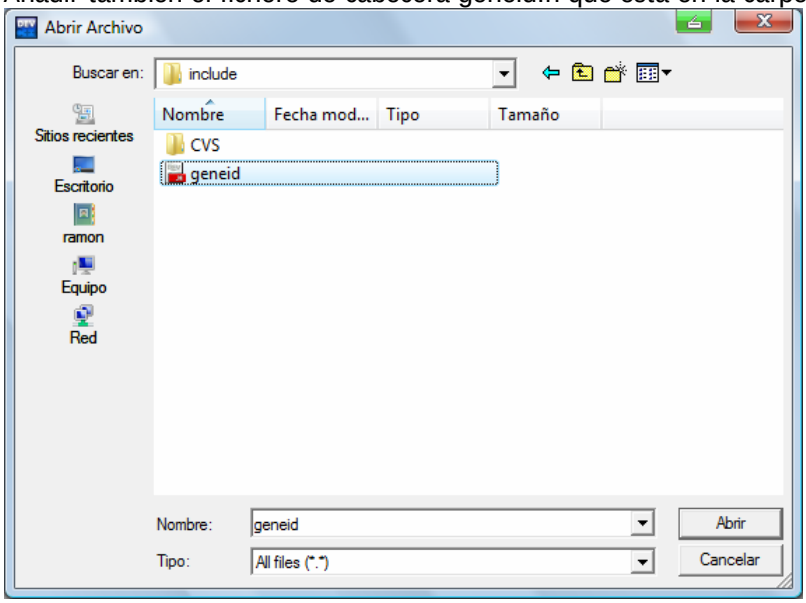
Quitar por defecto el modulo de c que incluye

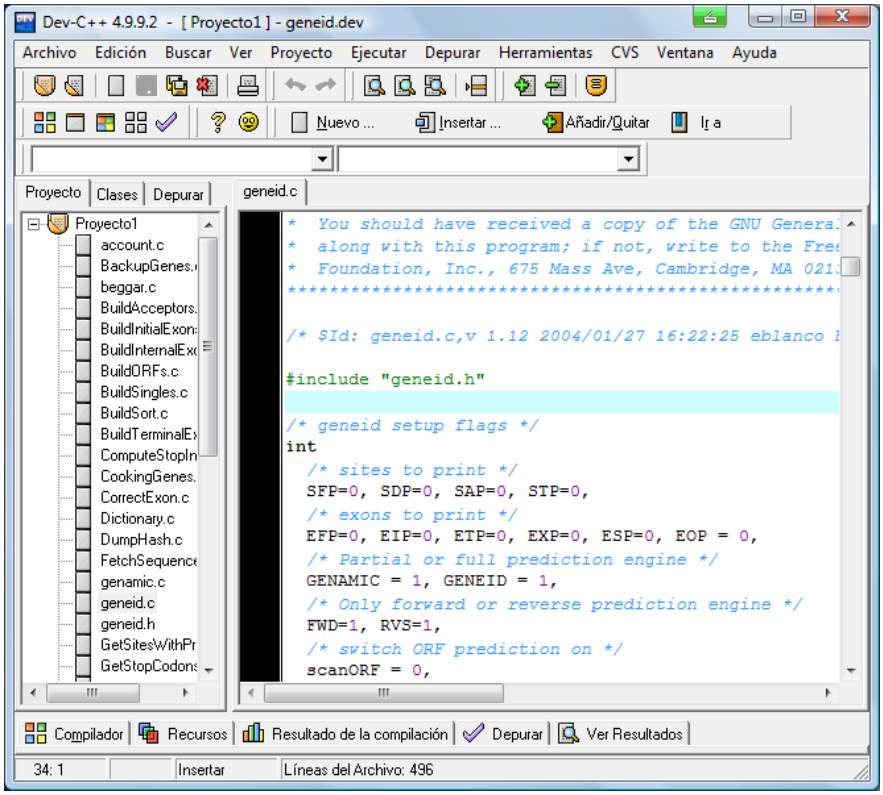


Añadir todos los archivos de geneid.



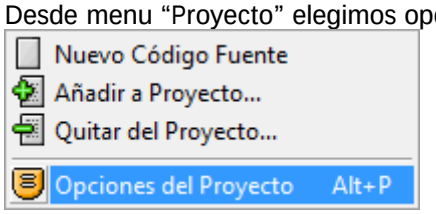
Añadir también el fichero de cabecera geneid.h que esta en la carpeta includes

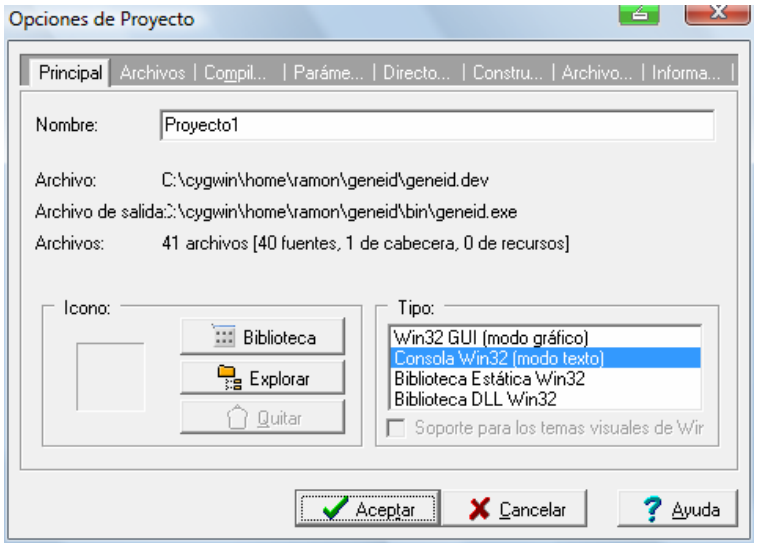


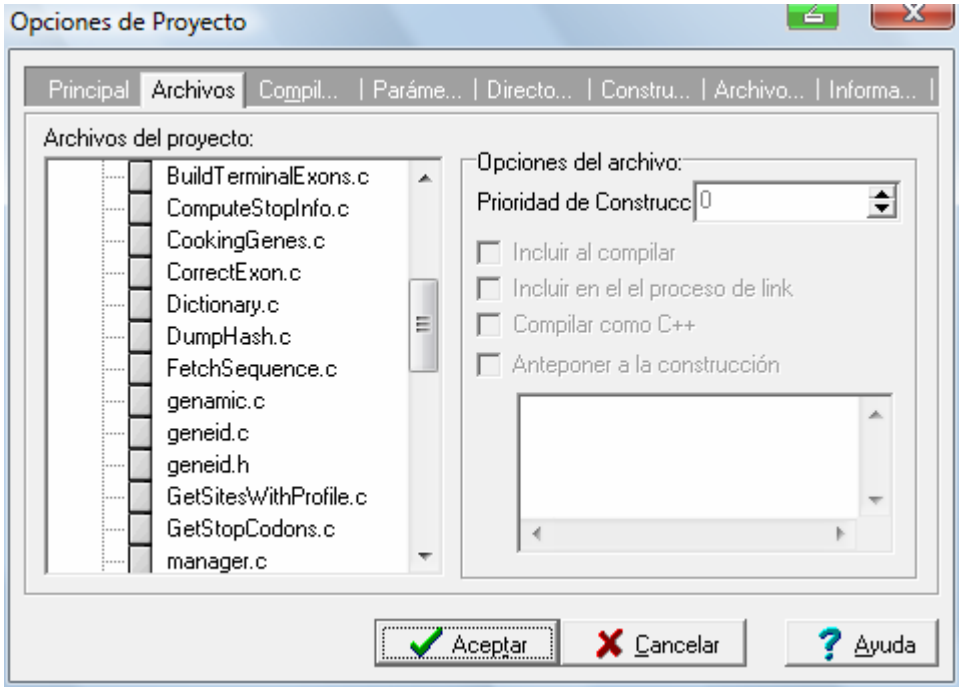


Configurar las opciones del proyecto

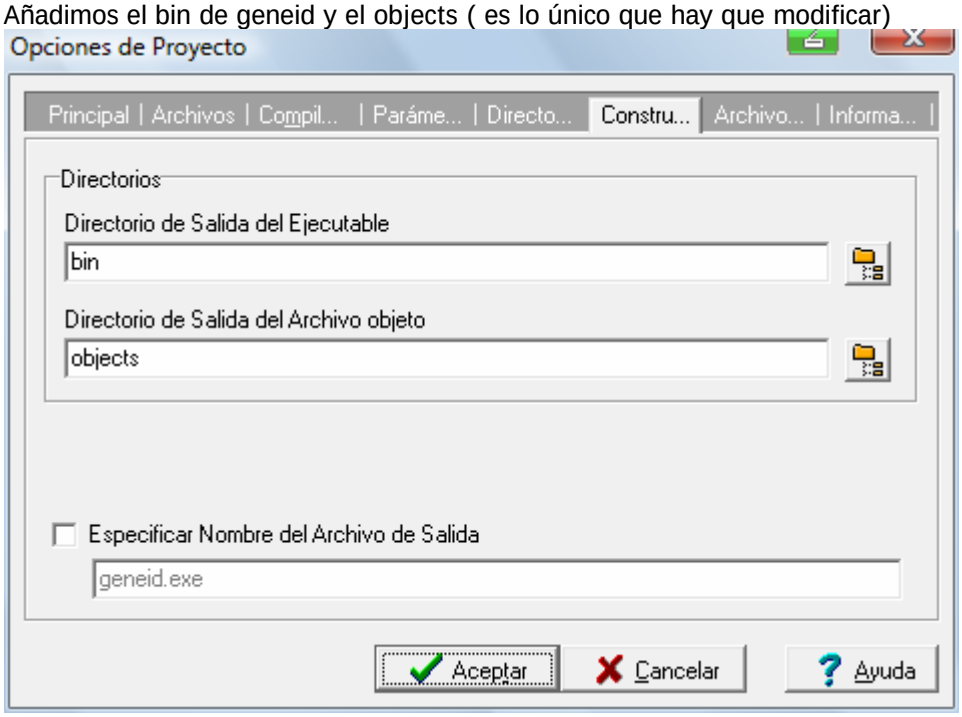
Desde menu “Proyecto” elegimos opciones del proyecto y modificamos lo que pone por defecto.





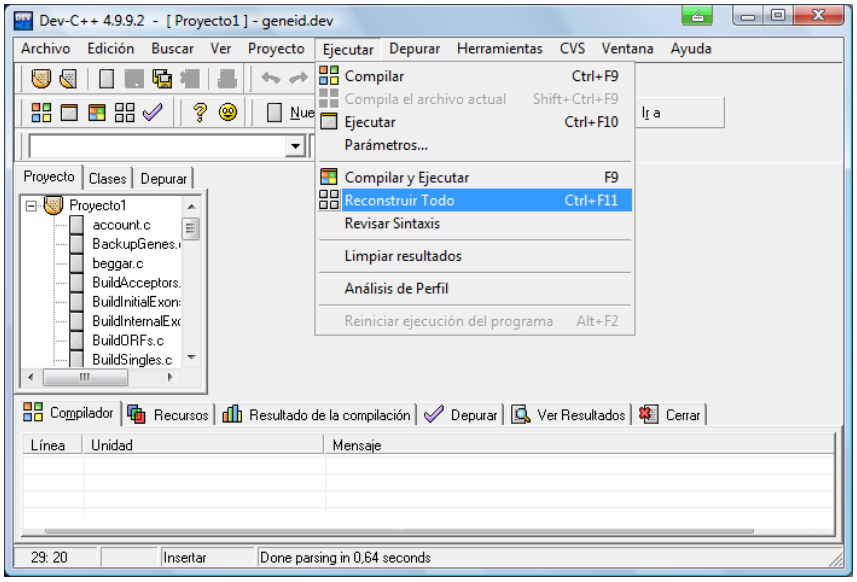


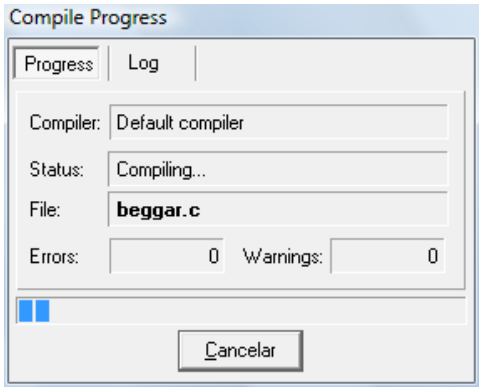
Añadimos el bin de geneid y el objects (es lo único que hay que modificar)



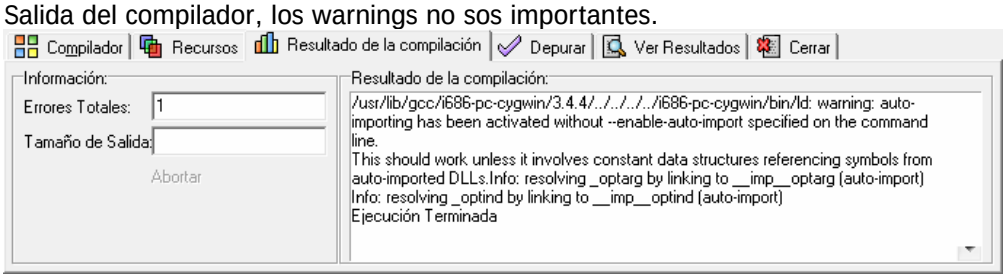
Lo demás lo dejamos igual. (principal, archivos, compilador, parámetros...

Reconstruir y compilar

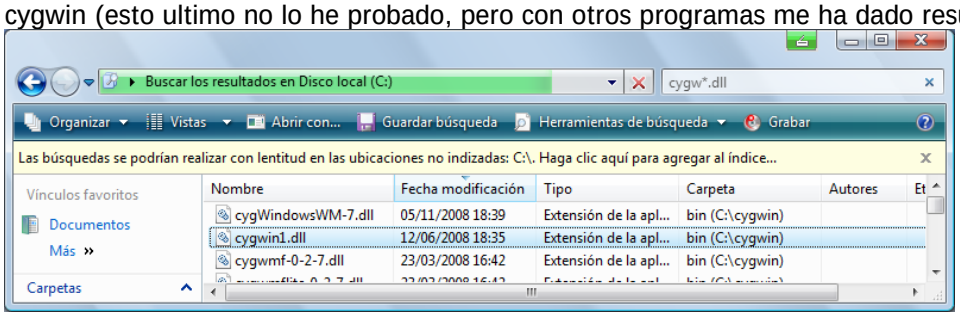




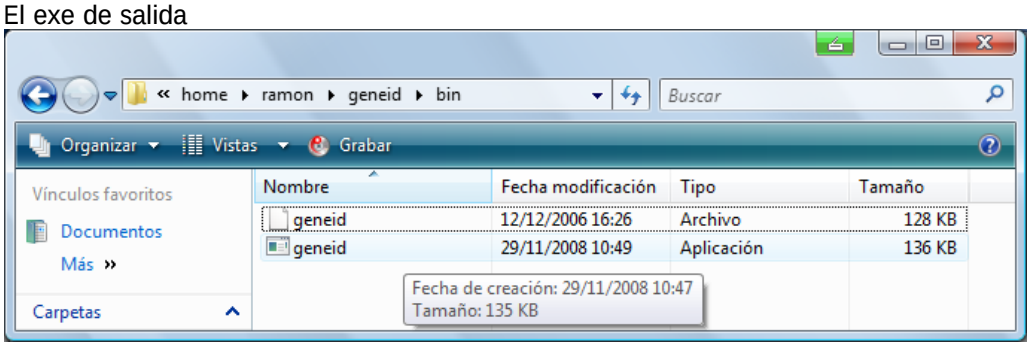
Salida del compilador, los warnings no sos importantes.



Si queremos que funcione fuera de cygwin u en otro PC hay que distribuir la librería cygwin1.dll junto con geneid.exe. O bien usar algún programa tipo “alloy” para insertarla en el código ensamblado de cygwin (esto ultimo no lo he probado, pero con otros programas me ha dado resultado)



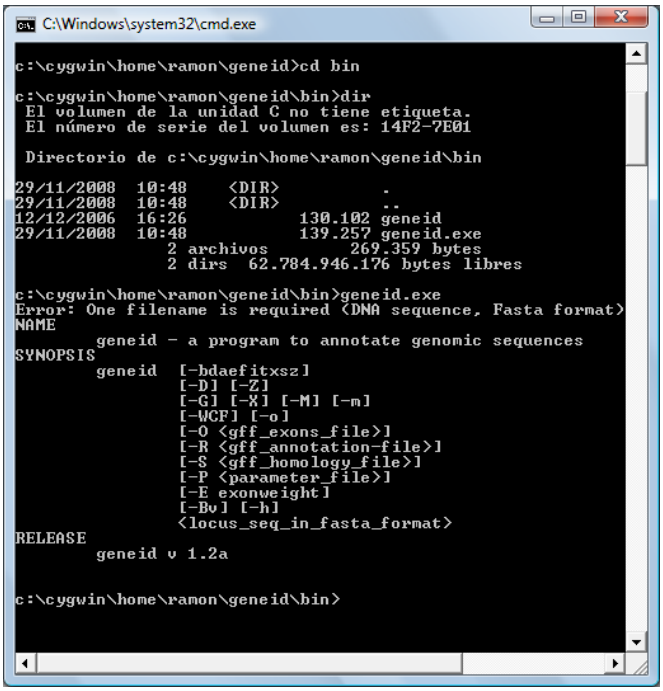
El exe de salida



(es un poco más grande porque incluye las cabeceras necesarias para rodar en windows)

Probando el geneid.exe

En el cmd de windows



En el cygwin como si fuera un programa unix (evidentemente, en un Linux no funcionará el exe)