gff2ps

— Version 0.96 —

Users Manual

Josep F. Abril Ferrando

Last Update
October 20, 2000

gff2ps v0.96

Converting genomic GFF-format annotations on PostScript plots.

License Terms

† **Current address:**

INSTITUT MUNICIPAL D'INVESTIGACIÓ MÈDICA
GENOME INFORMATICS GROUP
C/ Dr. Aiguader 80
08003 - Barcelona (SPAIN)
Phone: +34 93 221 10 09 ext 2016
Fax: +34 93 221 3237

e-mail: **jabril@imim.es**

GENOME INFORMATICS GROUP

Contents

List of Figures

List of Tables

Chapter 1 : Introduction

1.1 Description

gff2ps is a script program developed with the aim of converting gff-formated records into high quality one-dimensional plots in PostScript. Such plots may be useful for comparing genomic structures and to visualizing outputs from genome annotation programs. One of our goals is to write a program ready to handle the increasing amount of genomic information that is being obtained from genomic sequencing projects. Another important feature is that we have tried to separate record data from drawing features related to that data, avoiding to increase the complexity of gff-files with those drawing properties we will like to define for —such as feature color and so on—.

The program is not interactive as a Java applet and neither has a graphic user interface, we may add a more friendly interface in future developments. The main procedure must work basically as a filter which allows users to include our program within a command-line UNIX pipe. PostScript output can be displayed with ghostview or xpsview, although we often use the former, or sent directly to PostScript printing device (or other printers if you have the proper filters defined on your system), and can be included so easily into a LaTeX document too.

We hope that this program will be of usefulness in your analysis on genomic sequences.

1.2 About this manual

We wrote this manual having three blocks in mind: what is behind the code, how to work with it in a UNIX system, and how you can improve your plots using built-in variables. All three are very related among the others, because the input you are providing must fit a format, which is expected by the program, but modified by your choice in the custom file, and you must know how it is affected by command-line options, and so on. That's the reason why we suggest to read all three chapters in the same order that they appear, this will save you to spend time trying to solve an error that if you take care of our explanations perhaps it will not appeared.

The final appendices may help you as a reference guide when you start working with **gff2ps**. They summarize suggested GFF features, program options —and available parameters for those—, and command-line options.

POSTSCRIPT is a registered trademark of Adobe Systems Incorporated. UNIX is a trademark of AT&T/Bell Laboratories. LaTeX(by L.Lamport), LINUX (by L.Torwald), awk and bash (by GNU), gv (by J.Plass & T.O.Theisen), GhostView (by T.Theisen), Ghostscript (by Alladdin Enterprises), and **gff2ps** are all free software under GNU-GPL.

1.3 Acknowledgements

gff2ps was used to visualize the genomic data-sets submitted to ISMB'99 Tutorial "The challenge of annotating a complete eukaryotic genome: A case study in *Drosophila melanogaster*". We thank the opportunity given by Martin Reese and Michael Ashburner from the Berkeley Drosophila Genome Project. We appreciate the ISMB'99 organizing committee help very much for allowing us to show the three panels poster for the whole meeting.

We thank to Richard Bruskiewich, from Sanger Center and GFF-Specs Web Page curator, his helpful tips and comments on GFF-format. Also we thank to those people that have bet on GFF as data interchange format and those who have participated on the GFF mailing-list. We hope that GFF mailing-list will provide us a reference for GFF standard for a long time.

Finally thank to Moisés Burset, Genís Parra, Sergi Castellano and Enrique Blanco for their suggestions and their effort testing this program.

This work was supported by:

- Proyecto del Plan Nacional de I+D, BIO98-0443-C02-01, from the Ministerio de Educación y Ciencia (Spain).
- Beca de Formación en Investigación, BEFI 99/9345, from the Instituto de Salud Carlos III (Spain).

Chapter 2 : Basic Concepts

In this chapter we want to introduce you in GFF format definition as well as how **`gff2ps`** handles your GFF files. The main goal is that you will learn how "standard" GFF files that will run without problems to get nice plots are defined; you must know that **`gff2ps`**, when is checking input records, rejects all non "standard" GFF records (warning users on that, of course).

2.1 GFF philosophy

GFF Protocol Specification was initially proposed by Richard Durbin and David Haussler with amendments proposed by Lincoln Stein, Suzana Lewis, Anders Krogh and others. The GFF Specification is nowadays maintained at the Sanger Center by Richard Bruskievich[1].

We have developed **`gff2ps`** under **GFF Version 1** and **GFF Version 2** specifications, here we are going to summarize the basic aspects for it. You can obtain a more complete definition at Sanger Center GFF format page at:

`http://www.sanger.ac.uk/Software/GFF/gff.shtml`

and the full specs at:

`http://www.sanger.ac.uk/Software/GFF/GFF_Spec.shtml`

There is a mailing list to which you can send comments, enquiries, complaints, etc... about GFF. To be added to the mailing list you have to send a mail to **`Majordomo@sanger.ac.uk`** with the following command in the body of your email message: **`subscribe gff-list`**

2.1.1 GFF Format Definition:

GFF —General Feature Format (formerly called 'Gene Feature Finding')— is a format specification for describing genes and other features associated with genomic sequences and the transfer of feature information. A GFF record is an extension of a basic (name,start,end) tuple (or "NSE") that can be used to identify a substring of a biological sequence. (For example, the NSE (ChromosomeI,2000,3000) specifies the third kilobase of the sequence named "ChromosomeI"). Filename extension for GFF format has been defined as '.gff'.

GFF format is not conceived to be used for complete data management of the analysis and annotation of genomic sequences, there are other much powerful systems developed for that (as Acedb, Genotator, ...). This format is intended to be easy to parse and process by a variety of programs in different languages —i.e. Unix tools like grep, sort and simple perl and awk scripts—. So, GFF format has a record-based structure, where each feature is described on a single line, and line order is not relevant. In Version 2, all the fields of every single record must be separated by TAB characters ('\t'), revoking previous permissions to use arbitrary whitespaces as field delimiters. Table 2.1 placed below, shows some simple example records.

2.1.2 Fields Definition:

Fields are:

[1] `rbsk@sanger.ac.uk`

Table 2.1: The simplest GFF-format records.

SEQ1	netgene	splice5	172	173	0.94	+	.		
SEQ1	genie	sp5-20	163	182	2.3	+	.		
SEQ1	genie	sp5-10	168	177	2.1	+	.		
SEQ2	grail	ATG	17	19	2.1	-	0		
SEQ1	EMBL	atg	103	105	.	+	0	labl	#
SEQ1	EMBL	exon	103	172	.	+	0	labl	#
SEQ1	EMBL	splice5	172	173	.	+	.	.	#

<*seqname*> <*source*> <*feature*> <*start*> <*end*> <*score*> <*strand*> <*frame*> [*group*] [...]

<**seqname**> The name of the sequence. Having an explicit sequence name allows a feature file to be prepared for a data set of multiple sequences. You can use the identifier of the sequence in an accompanying file containing the sequence nucleotides string, or the identifier for a sequence in a public database —as EMBL/Genbank/DDBJ accession number—.

<**source**> The source of this feature. This field will normally be used to indicate the program making the prediction, or if it comes from public database annotation, or is experimentally verified, etc...

<**feature**> The feature type name. As you can use other features, it would be desirable to have a Standard Table for common features. For this standard table has been proposed to fall back on the international public standards for genomic database feature annotation, specifically, the DDBJ/EMBL/GenBank feature table[2]. Some of the most used terms in genomics from that table are summarized at Appendix A.

<**start**>, <**end**> Integers. <start> must be less than or equal to <end>, so reverse strand coordinates must be defined in forward coords. In GFF Version 1 sequence numbering starts at 1, so these numbers should be between 1 and the length of the relevant sequence, inclusive. Version 2 condones values of <start> and <end> that extend outside the reference sequence.

<**score**> A floating point value. When there is no score you should use '.'.

<**strand**> One of '+', '-' or '.'. '.' should be used when strand is not relevant.

<**frame**> One of '0', '1', '2' or '.'. '0' indicates that the specified region is in frame, i.e. that its first base corresponds to the first base of a codon. '1' indicates that there is one extra base, i.e. that the second base of the region corresponds to the first base of a codon, and '2' means that the third base of the region is the first base of a codon. If the strand is '-', then the first base of the region is value of <end>, because the corresponding coding region will run from <end> to <start> on the reverse strand. As with <strand>, if the frame is not relevant then set <frame> to '.'. It has been pointed out that "phase" might be a better descriptor than "frame" for this field.

[group] An optional string-valued field that can be used as a name to group together a set of records. Typical uses include to group the introns and exons in one gene prediction (or experimentally

[2]See the DDBJ/EMBL/GenBank feature key table definition at:
http://www.ebi.ac.uk/embl/Documentation/FT_definitions/feature_table.html

verified gene structure). In Version 2, group must be defined within a Tag-Value pair. Tags must be standard identifiers ([A-Za-z][A-Za-z0-9_]∗). Free text values must be quoted within double quotes. Examples for group field can be found in table 2.2. Standard table for Group Tag Identifiers has not yet been completely formalized, however a useful constraint is that they are equivalent, where appropriate, to DDBJ/EMBL/GenBank feature 'qualifiers' of given features[3].

Table 2.2: Grouping GFF records.

Simple Group Names (GFF Version 1)								
CETBB	search	cds	2189	2884	.	+	.	125
rt2202	predict	gene	1289	12852	64.07	−	.	trypsin
MMPROT	blast	similarity	32727	32740	1.6e-23	+	1	"RNA polymerases"
Tag−Value Group Names (GFF Version 2)								
jj_lk2	finder	cds	6718	7051	.	−	.	Transcript "1"
dJ102G20	GD_mRNA	exon	7105	7201	.	−	2	Sequence "dJ102G20.C1.1"
seq1	BLASTX	similarity	101	235	87.1	+	0	Target "HBA_HUMAN" 11 55

Comments Comments are allowed starting with character '#', everything following '#' until the end of the line is ignored. Effectively this can be used in two ways: at the beginning of the line to make the whole line a comment, or the comment could come after all the required fields on the line.

Meta Information You can define optionally a number of special comment lines for meta information at the top of your gff file with '##'. Current proposed '##' lines are:

##gff-version {version}
GFF version, current version is 2.

##source-version {source} {version}
You can record program or package version generated the data in this file.

##date {date}
The date the file was made, or perhaps when prediction programs were run. Use of astronomical format is recommended (1997-11-08 for 8th November 1997), first because this sort properly, and second to avoid any US/European bias.

##DNA {seqname}
##acggctcggattggcgctggatgatagatcagacgac
##...
##end-DNA
To give a DNA sequence. Several people have pointed out that it may be convenient to include the sequence in the file. It should not become mandatory to do so. Often the seqname will be a well-known identifier, and the sequence can easily be retrieved from a database, or an accompanying file.

[3] See the EMBL feature and qualifiers description at:
http://www3.ebi.ac.uk/Services/WebFeat/

##sequence-region {seqname} {start} {end}
To indicate that this file only contains entries for the specified subregion of a sequence.

Table 2.3: Some remarks on GFF Standard Format Version 2.

- ▷ Intended to be easy to parse and process.
- ▷ Field separator must be a TAB character ('\t').
- ▷ Fields must not include whitespace.
- ▷ <start> must be lower or equal than <end>.
- ▷ When there is no <score> you should use '.'.
- ▷ When <strand> is not relevant you should use '.'.
- ▷ Available <frames> are '.', '0', '1' and '2'.
- ▷ Tag-Value pairs accepted for <group> field.
  - ◦ Group tag must be a standard identifier ([A-Za-z][A-Za-z0-9_]$*$).
  - ◦ Free text values must be quoted within double quotes.

2.2 `gff2ps` philosophy

The programming philosophy underlying **`gff2ps`** can be summarized onto these points:

- We want to generate comprehensive plots of all 'GFF-able' features in order to compare genomic sequences from different sources. Although developed initially to display features annotated from different sources on a single sequence, it can also be used for displaying annotations from one (or more) sources on a number of sequences. This can be useful, for instance to compare the genomic structure of different sequences.

- **`gff2ps`** can parse Version 1 and Version 2 GFF-formatted records, records that are not compliant with GFF-format are discarded warning user afterwards. Field separator and group field were defined slightly different from one version to the other, here it is explained how **`gff2ps`** deals with those differences. If records from input GFF-files contain tabulators as field separator program assumes that record is GFF Version 2 formatted, else if it finds blank-spaces as field separator then switches to Version 1.
  Groups must be defined as a 'tag-value' pair in GFF Version 2, where 'tag' must be a standard identifier ([A-Za-z][A-Za-z0-9_]$*$) and 'value' must be a free-text string enclosed between double-quotes (' `".*"` '), for example '`target "HS new gene"`' fits that group format. Other group formats force program to switch to GFF Version 1: if there are more than nine fields it tries to find the 'tag-value' pattern, else assumes that the ninth field is the group name as a quoted or not free-text string. See figure 2.2 for group formats than can be processed by **`gff2ps`**.

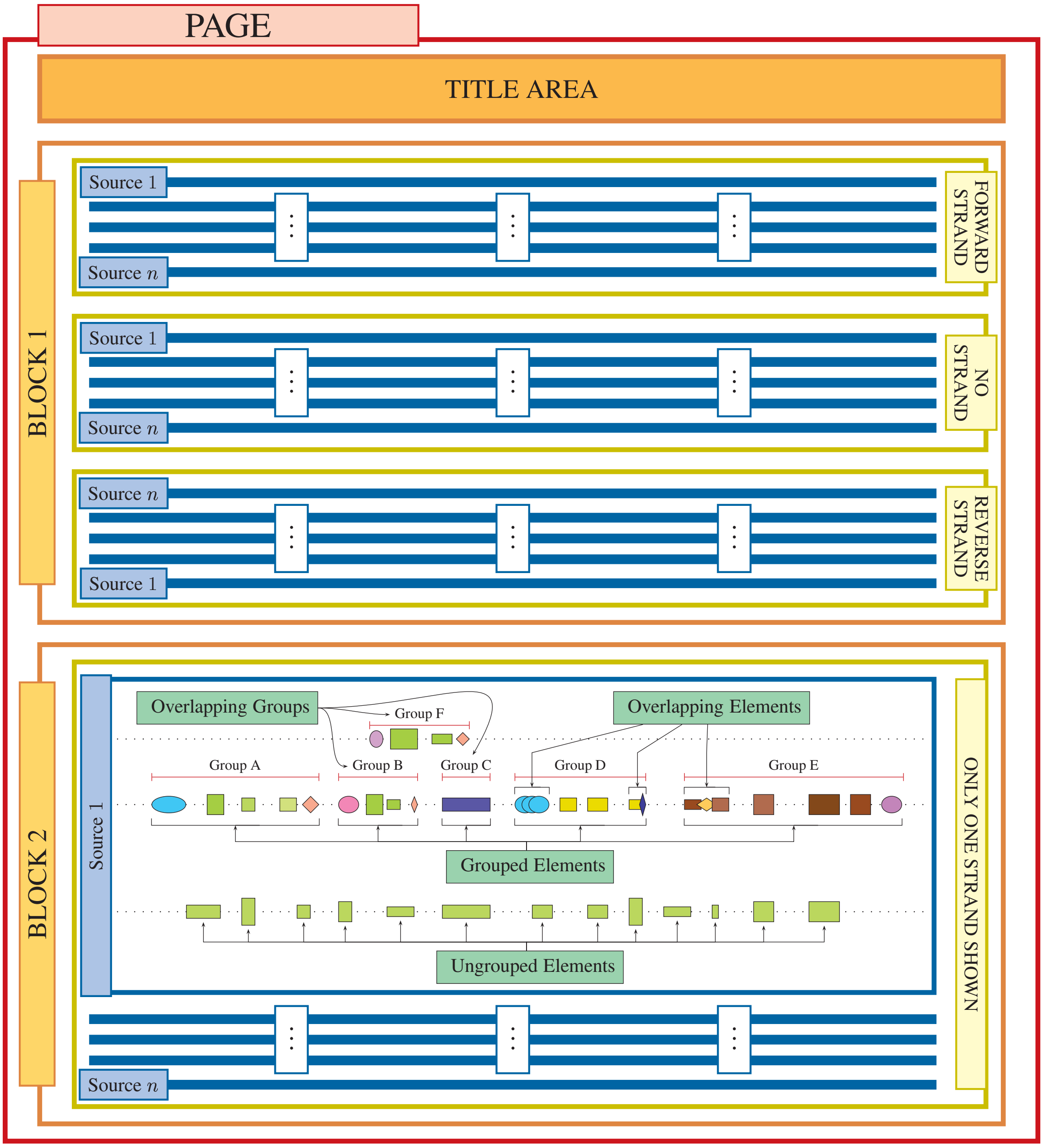


Figure 2.1: Here, you can observe in upper Block how strands are distributed on block area, and also how sources have reverse order in reverse strand. You can disable visualization for any strand and place one or more blocks per page. Imagine that lower Block has the same distribution as upper one, but the figure is only showing one zoomed strand. That strand also focuses on one source track, what is represented on it, and how we represent grouped and un-grouped elements generated from GFF-features. There is also shown overlapped elements and groups, each of them can be treated in different ways by **`gff2ps`**.

- The program must be easy to use, all its parameters are set by default inside the program. But must be easy to change plot options which can be modified by a default custom file, also by a working custom file (smaller than default file and provided to introduce small changes for single plots), and some of them from command-line. The main goal is to define a system

GFF Pattern		Group Examples

`[1-8] .* [...]`	⇒	`123`
		`DMSELE.1`
		`Clone_33223`

`[1-8] ".*" [...]`	⇒	`"123"`
		`"DMSELE.1"`
		`"Clone_33223"`
		`"Brain K+ Channel"`

`[1-8] [A-Za-z][A-Za-z0-9_]* ".*" [...]`	⇒	`target "123"`
		`SIMILARITY "DMSELE.1"`
		`label "Clone_33223"`
		`Putative_Protein "Brain K+ Channel"`

Figure 2.2: Group formats read by **`gff2ps`**. '`[1-8]`' represents the first eight gff-fields of each GFF-record. '[...]' corresponds to extra fields that are not used by our program.

in which can be easily added new options or redefine old ones. Another issue is that configuration files are plain-ASCII text, so they can be edited with small and simple text editors. To know more about this issue you can read section 4.1 and 4.2. The program can work in background or used in a UNIX pipeline working as a filter (section 3.2).

- Source order from input gff-file is preserved when reading those files, it means that you can easily switch source order in your plot swapping the order of the input files. Sources are shown in plots giving a mirroring symmetry axes for strands forward and reverse, so forward sources are shown by its ordering from top to bottom and reverse sources are shown counterwise, while records without a defined strand are placed in the plot area between the two strands areas.

- User-defined custom files can handle regular expressions, allowing us to define any attribute variable for multiple similar features —in GFF-elements, group or source blocks— in one line, to know about this feature you can look at section 4.2.2.

- Some of the previous items leads to hierarchical plots, in which Pages are the highest element, and Blocks are defined within them —this feature allows you to get multiple vertical and/or horizontal pages, also pages with multiple blocks—. Inside Blocks may appear any Strand —forward, reverse, or no-frame (defined in GFF-records as '.')—. Each Strand presents each Source as one plot line or more —if you want to display overlapping groups in the same line and/or in different lines or if you prefer to split data-sets between grouped and un-grouped features—, meanwhile Groups belonging to them define features for displaying sets of GFF-features, which are the basic plot elements (schemed in figure 2.1). Page number and Blocks per page are set as a Layout variables (see section 4.1.1), whereas the rest of other elements are defined in specific fields from GFF-files records: Sources are named at second field, GFF-features came from third field, Strands from field seven, and Groups from ninth (see section 2.1.1 and tables 2.1 and 2.2). Start, End, Score and Frame are defined into GFF-features as plot attributes.

- You can switch on/off visualization of overlapping groups in several lines for same source track —option by default—, or you can fit them into a single track. **`gff2ps`** minimizes the number of lines needed for that when displaying overlapping groups in different lines.

Individual non-grouped GFF-elements are treated as a one element group, which allows you to display also individual elements without overlapping. If you choose to print in a single line, you can also define layers for each set of overlapping GFF-features, maybe 'exons' at top and 'cds' at bottom, enhancing viewing for any of the elements. See sections 4.1.2 and 4.1.3.

- You must remember that **`gff2ps`** converts all upper-case characters for **features** to lowercase. In order to prevent that one user had defined 'Exon', another 'exon' and other 'EXON', our program convert them to 'exon'.

- Scores control feature width, but in order to prevent problems when working with data-sets provided by different programs they are re-scaled for each source, using maximum and minimum values as a score range within all scores are re-calculated. Default score is set to maximum value for each source (when parsing gff-files and a record contains a '.' in Score field). Further information can be found in section 4.1.4.

- We have defined three block areas where are displayed Strands, you can switch on/off any of them and visualize one up to three strands. Forward strand ('+') is always shown at upper area, while reverse strand is always shown at lower area. When strand is not defined ('.') elements are placed in the central area, between forward and reverse areas. Source order is preserved from input files, in forward strand sources follows that ordering up to down, also in the no-strand area, but in reverse strand are shown down to up, so you have a horizontal symmetry axes between forward/no-strand and reverse strand.

- Features for which frame is specified are plotted using a two color code schema. The upstream half of the graphical element representing the frame of feature and the downstream half the complement modulus three of its remainder. This is useful to check frame consistency between adjacent features (for instance, predicted exons). Two adjacent features are frame-compatible when the color of the downstream half of the upstream feature matches the color of the upstream half of the downstream feature. This two-color code schema, however, is only meaningful when the frame has been defined relative to the feature, and not relative to the sequence. We have defined four independent frames in **`gff2ps`**('.', '0', '1', and '2'), that is used by a coloring procedure for visualizing frame and remainder within shapes. Colors for frames are also independently defined from feature color definition, and can be customized too. The complement modulus three for "remainder" is calculated by our program following this formula:

$$\textit{``Remainder''} = (3 - (\textit{End} - (\textit{Start} + \textit{Frame}) + 1) \bmod 3) \bmod 3$$

- Score vectors are shown as score-dependent color gradients and are read from one line which contain such vectors in those fields beyond group field. We will implement in next version procedures for showing them as continuous or discrete functions, spikes, and so on... You must use the following format for the group and the following fields (next version will work also with multiple GFF score-feature single-records) :

[Fields 1 to 8] Tag "Value" **score**; **Window** *window*; **Step** *step*; **Scores** *score* [. . .] *score*

Chapter 3 : Using gff2ps

In this chapter it is explained how you can set system variables in order to start working with **gff2ps**. This program was designed to work under UNIX and has been tested under Irix, Solaris and Linux. In table 3.1 you can see the program versions with which we have worked. **gff2ps** has three inner modules: the shell script, the GNU awk script, and the PostScript prologue code. This prologue contains all procedure sets we have written to obtain the PostScript plots and it is embedded into '`gff2ps`'. This is a Bourne shell script —using *sh* or *bash* (systems under Linux have a link for *sh* to *bash*)–, that handles with command-line options, checks if given files exist and pass them to the GNU awk script. This loads data records and custom definitions generating the full PostScript output. You can visualize that with a PostScript viewer —like ghostview, xpsview— or send to a PostScript printer to obtain a hardcopy.

The main difference from older versions is that GNU awk script is now included within the shell script to facilitate installation of our program.

3.1 Installation tips

Once you have downloaded **gff2ps** compressed tarball ('.tar.gz'), you must decompress files using the following commands:

```
[cshell]$ gunzip -c <file> | tar xvf -
```

If you are working on Linux system, you can try with:

```
[cshell]$ tar zxvf <file>
```

It will create a 'gff2ps/' directory containing a 'README' file, this manual, the **gff2ps** scripts and a subdirectory with some PostScript examples[1], as is shown in figure 3.1.

You can move those files to another directory if you want. Make sure too that the main program has executable permissions.

```
[cshell]$ ls -alF gff2ps
-rwxr-r- 1 jabril users 50360 Sep 13 17:42 gff2ps*
```

If the program has not execution permissions (boldface highlighted on above line) you can change it with:

```
[cshell]$ chmod u+x gff2ps
```

After that, you may define the following environmental variables, although none is mandatory:

[1] You can visualize the examples and get new ones at:
http://www1.imim.es/~jabril/GFFTOOLS/GFF2PS-Snapshots.html

```
gff2ps/
    ├──── gff2ps
    ├──── README
    ├──── GNU_GPL
    ├──── MANUAL_GFF2PS.ps
    └───── examples/
                  ├──── Default.tar.gz
                  ├──── 1bx1p_long.tar.gz
                  ├──── 2bx1p.tar.gz
                  ├──── 3bx1p.tar.gz
                  └──── align_and_shapes.tar.gz
```

Figure 3.1: Tree directory for gff2ps tarball.

- ◇ **gff2ps** needs to write few temporary files in a directory with read and write permissions for current user. Default temporary directory path is set to "/tmp/" but you can assign a different temporary directory path using the variable 'GFF2PS_TMP'.
- ◇ You can specify the path where **gff2ps** can find the default files with the shell variable 'GFF2PS_CFDIR'. Default value is path where you are running **gff2ps**.
- ◇ You can also define the default custom filename you will like with the variable 'GFF2PS_CUSTOMFILE', program default filename for custom file is '$GFF2PS_CFDIR/.gff2psrc'.
- ◇ If you have several versions of GNU awk installed in your system —we recommend version 3.0 or greater if you are going to work with large datasets—, or you do not have GNU awk in your path, you can provide the path for that GNU awk that needs **gff2ps**. The program can read a shell variable for the GNU awk path named 'GAWK_DIR', and it will try to run GNU awk as ' "$GAWK_DIR"gawk '. By default, **gff2ps** run GNU awk as 'gawk' expecting you can acceed to the interpreter from your path.

Using a Bourne shell (e.g.sh or bash):

```
[bshell]$ export GFF2PS_CFDIR="path"
[bshell]$ export GFF2PS_CUSTOMFILE="file_name"
[bshell]$ export GFF2PS_TMP="path"
[bshell]$ export GAWK_DIR="path"
```

Using a C-Shell (e.g. csh or tcsh):

```
[cshell]$ setenv GFF2PS_CFDIR "path"
[cshell]$ setenv GFF2PS_CUSTOMFILE "file_name"
[cshell]$ setenv GFF2PS_TMP "path"
[cshell]$ setenv GAWK_DIR "path"
```

Now your system is ready to work with **gff2ps**. Just another tip before going further, if the program is not working yet and you get an error message like:

```
/bin/sh:  Command not found.
```

then you do not have the Bourne shell in the standard directory, and **gff2ps** could not work. The only way to solve this problem is changing the first line of the script. From your command-line, type:

[*bshell*]$ which sh

or

[*bshell*]$ which bash ('bash' is more powerful than 'sh')

You will get the directory from which you can run the Bourne or the bash shells (in case you have many directories choose then one for the latest shell version). After this search, you can modify the first line from the **gff2ps** script on any ASCII text editor (like vi or emacs), adapting our script to your needs. So, as example, imagine you have looked for the shell path in your system, by typing:

```
[bshell]$ which bash
/usr/bin/bash
/bin/bash
```

and the latest version is on '`/usr/bin/bash`'. Replace the first line on **gff2ps**, that looks like:

```
#!/bin/sh
```

by

```
#!/usr/bin/bash
```

Now, you have tailored **gff2ps** to work in your system.

3.2 Using gff2ps on command-line

The basic idea is that our program works as a filter having an input, an output, and an error channel. In figure 3.2 you can have an idea of that, but if you are not used to work on UNIX it is better for you to take a look into any introductory UNIX manual for a reference on file redirections and pipelines[2,3,4,5].

The simplest command line for **gff2ps** is:

**[*cshell*]$** gff2ps *gff_file*$_1$ ...*gff_file*$_n$

Here, input is taken from n 'GFF files' —you can pass from one to n different files or you can concatenate them in a unique input file—, but it can be passed to standard input from a "pipe" command ('|'). Output is shown by standard output. All error, warning and report messages are redirected to standard error. The problem that arise is that default standard output and default

[2] D. Gilly. "UNIX in a Nutshell: System V Edition" O'Reilly & Associates Inc. 1994 (2nd ed.)
[3] L.J. Arthur, T. Burns. "UNIX Shell Programming" John Wiley & Sons Inc. 1997 (4th ed.)
[4] G. Anderson, P. Anderson. "The UNIX C Shell Field Guide." Prentice-Hall Inc. 1986 (2nd ed.).
[5] C. Newham, B. Rosenblatt. "Learning the Bash Shell" O'Reilly & Associates Inc. 1998 (2nd ed.).

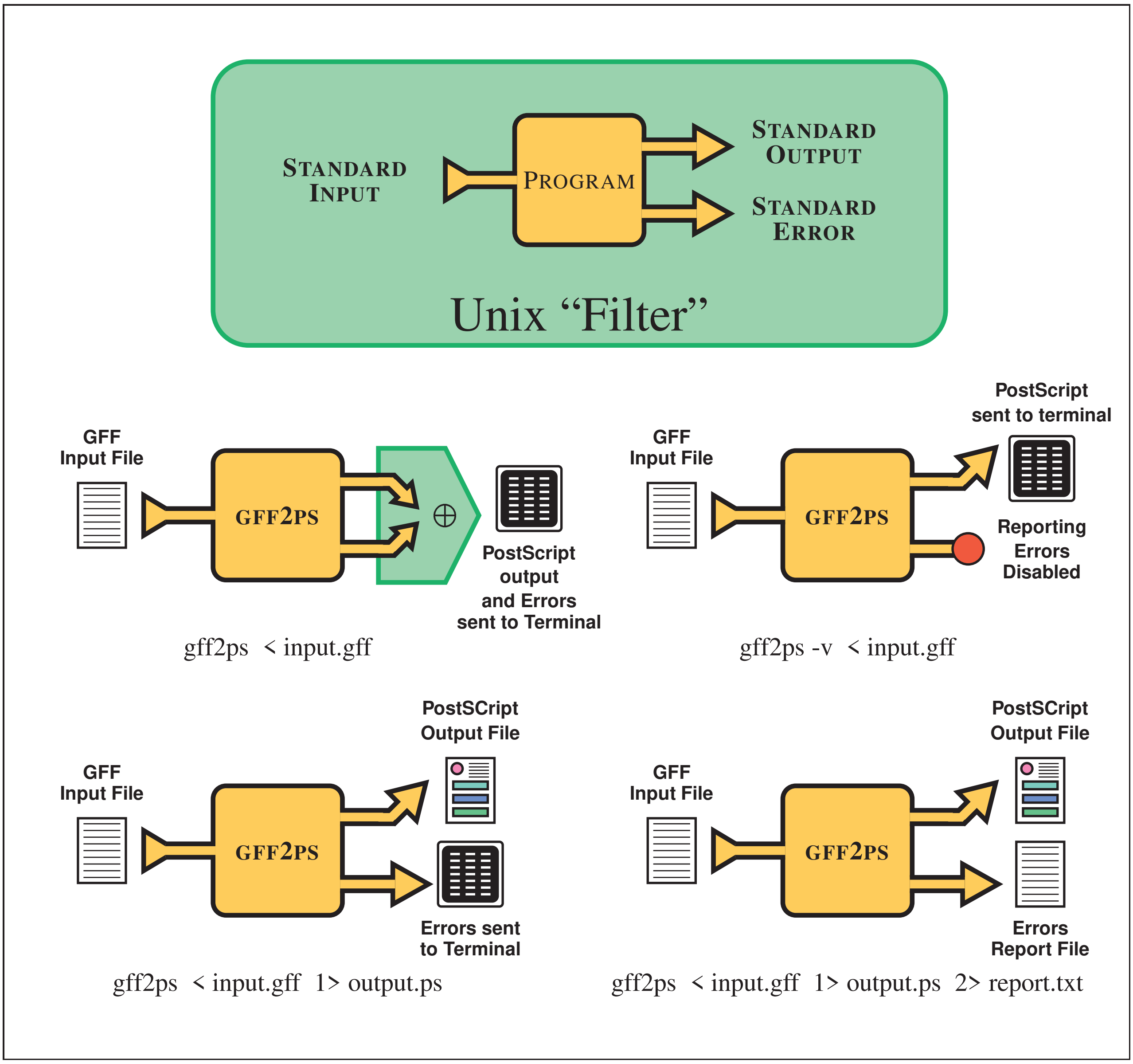


Figure 3.2: UNIX pipes and **gff2ps**. Redirecting Standard Error and Standard Output to same file is not shown but is not useful because if you try to print as a PostScript file then you get a printer error (redirections are shown in bash command-line format).

standard error are dumped together to console; which means that you will get program warnings mixed with PostScript output, and that it will not work when you try to print.

3.2.1 Generation of report files

How can we solve that? There are two solutions: on the one hand, if you are not interested on standard error output you can disable error reporting with ‘-v’ option (**gff2ps** quiet mode); on the other hand, if you are interested to have both, you can redirect PostScript output from standard output to a file. First option is useful when including our program in a shell script; second is better when you are working in command-line because you can detect on the fly if something is wrong.

Both ways are illustrated here:

**[*cshell*]$** gff2ps *gff_file*$_1$ ...*gff_file*$_n$ >*output.ps*

Writes output to *output.ps*.

**[*cshell*]$** cat *gff_file*$_1$ ...*gff_file*$_n$ | gff2ps -v -- - | ghostview -

In this example (also very simple), input come from 'cat' output redirected to standard input (defined by '-') and standard output is redirected to 'ghostview'. Note that there is a double hyphen ('--') following option '-v', this notify to shell-script 'getopts' function that there are no options left, for further info about that see section 4.3. By default, only errors and warnings are shown by standard error, you can disable all of them as I've shown you above or you can enable a full report —that will be useful to report bugs as I explain in the next section— with '-V' option (note that is an upper-case here). When reporting bugs you have to attach a report file that will be obtained redirecting standard error to a file, you can see examples of this on lines below, for c-shell and bash:

**[*cshell*]$** (gff2ps -V -- *gff_file*$_1$ ...*gff_file*$_n$ >*output.ps*) >&*output.rpt*
**[*bshell*]$** gff2ps -V -- *gff_file*$_1$ ...*gff_file*$_n$ >*output.ps* 2>*output.rpt*

3.2.2 Command-line help

To know about available Command-line Options, you can type:

[*cshell*]$ gff2ps -h

Option '-h' sends to terminal a short description on each **gff2ps** options, also gives some extra information about color names, and environment variables. There is another help option '-H <*option*>' that gives you only the help line for that option, and its useful when you are not sure about what any option is doing. An example:

```
[cshell]$ gff2ps -H v
  gff2ps :  Option Definition Help.
    -V Verbose mode, a full report is sent to standard error
       (default is only send Warnings).
    -v Silent mode:  Disable all warnings, no messages sent
       to standard error.
```

Also you can get a pretty print for the options listed by help in appendix B.

3.3 Reporting Bugs

If you need help after reading this manual pages, detect a bug, or you have any suggestions, you are free to send an e-mail to the author: jabril@imim.es. In order to organize incoming mails, you can help us defining as **Subject** of your e-mail: "GFF2PS - HELP" if you need any clarification;

"GFF2PS - BUG REPORT" if you find any bug or something is not plotted properly; and "GFF2PS - SUGGESTION" if you think that something could be improved or you want new ideas to be included. You will get a reply as soon as possible. Also when reporting a bug, it will be useful to attach a report file from **gff2ps** —see previous section on how to get it— and program versions that you have installed on your system —like those reported in table 3.1—.

If you have problems when printing **gff2ps** output or something is not plotted properly, first check printer settings on your system. Try printing another PostScript file to test if is a printer problem or from our output. Our PostScript code has been tested on PostScript Level 2 and on Level 3 printers, but, if you have enough memory in your printer it has to work in a Level 1 because we have tried to avoid specific Level instructions when we wrote that code. It's also possible to print to a non PostScript printer if you are using Ghostview or similar to visualize the results, and if you get in trouble, please make sure that is not a problem of filters defined for those printers before reporting a printing bug. If all of this does not work then send a bug report mail, specifying which printer you have, its specs, which PostScript Level has that printer, any printing error message from the system, and do not forget to attach a file containing the PostScript output you have obtained, the one is giving you such problems.

Red Hat LINUX	IRIX	SOLARIS
GNU Awk 3.0.3	GNU Awk 3.0.3	GNU Awk 3.0.3
GNU bash 1.14.7(1)	GNU bash, version 1.14.2(3)	GNU bash, version 2.01.0(1)
gv 3.5.8	Ghostview v.1.5	Ghostview v.1.5
GNU Ghostscript 4.03	Ghostscript 2.6.2	Aladdin Ghostscript 3.33

Table 3.1: **gff2ps** has been tested on those Unix systems, working with shown program versions.

Chapter 4 : Setting Output

In this section we should learn first of all which options are available on **gff2ps** to modify attributes for plot features from custom files. Those features are divided into four groups of variables: page layout, GFF-elements, groups and sources. The two reasons for such classification is the way that custom files are processed by our program, and what each group in the final plot represents. The second reason is the most important and was the one which lead us to read and process them the way they are by our program. Layout Features are involved in global page layout, variables from this block affect page format and number, title area, how many blocks are displayed, strands shown. Features related to GFF-elements are relative to how they are drawn, their shapes, fill color, and so on. Group features define how to represent grouping for sets of GFF-features, and permit to redefine the attributes of GFF-features contained within a Group. Finally, Source features are controlling the setting for plot rows, understanding them as a set of groups and features obtained from one source. You might found that hierarchy so strange, but working with genomic data you can easily see that sources represent the methods —gene prediction programs, blast matches, etc. . . — from which you get the genomic data, and you may be interested on comparing them. In that sense Groups can be defined as "genes" and GFF-features as the genomic elements that compose those genes —such as exons, introns, and so on—.

Once we know which options are available, we should discuss about what happens whether you have defined the same variable on different sections (such feature_color) and which variable is used by **gff2ps** in case of conflict between them. We also should consider how **gff2ps** manages customization files and how can you take advantage from regular expressions to simplify your custom files when working with large GFF-formatted data-sets. Then we should know what options are available from command-line and how they affect to custom files previous settings. There are also a hierarchy on how options are processed by **gff2ps**; when you are running the program, the first step is generating a defaults array for all options, the next step is processing your GFF-data to extract GFF-features, Groups and Sources contained on it. Following steps are reading Default Custom File if it exists and, after that, the Extra Custom File if user provides it; plot attributes for every element —GFF-features, Groups, Sources and Layout— are redefined from program defaults to custom files settings. Before PostScript output generation, command-line options introduce last modifications on plot attributes array, which is now containing information on how plot attributes are going to be drawn. As you can see, defaults from program are overridden by Default Custom File, those are overridden by Extra Custom File and finally Command-Line Options has the last word on plot attributes.

4.1 Custom File Variables Definition

Here we are going to define how plot attributes for Custom Files must be defined to be read by our program. There are only three fields on each variable definition: plot element, variable name, and value for that variable. This is not true for Layout Features that only need two fields, plot element is not defined on them because all belong to same element, named "Layout" of course. Generic format for attribute definition is:

For Layout features:

<*variable_name*>=<*value*>

For GFF-element, Group and Source features:

<*plot_element*>::<*variable_name*>=<*value*>

Notice that field separator between <*plot_element*> and <*variable_name*> is a double colon ('::') and <*value*> assignment is made by equal sign ('=').

Plot attributes, as we have explained, are grouped in four types: those relatives to global attributes and those related to GFF-features, Groups or Sources. Section 4.1.5 explains how **gff2ps** solves conflicts between variables from defaults, custom files and command-line. You will find a list of all available variable names with a short explanation of what they do and available values for them in Appendix B. In the following subsections you will find a description for all the available custom-file variables. Header on each description contains the variable name as it will be written on the custom files, applying previous variable definition format, on the left side and default values on the right one. When more than one value are available for those variables, it is represented by a '|', meaning that you can assign any of those values (but only one) to the variable. Other values are explained on variable description. References to value tables are included when there are so many.

In the following subsections, <*BOOLEAN*> means that you can pass to the variable any of those values: '1' or '0', 'on' or 'off', 'yes' or 'no' ('y' or 'n'), 'true' or 'false' ('t' or 'f'); either in lower-case or capital letters. Other values are set to '0'.

4.1.1 Layout Features

As we have stated before, layout variables don't need to be assigned to a plot element because they share same element, and so they have a slightly shorter variable definition which miss the first field —describing plot element— and the double colon. Here there are the Layout Features listed:

page_size=<*page_format*> **Default= a4**

Available values for <*page_format*> are shown in table C on 'Page Format' column.

page_bbox=<*page_format,page_width,page_height*> **Default= auto,0,0**

Default value force program to use previous variable definition, which sets pre-defined widths and lengths for many page formats. Do not edit unless you want to define a new page format with your own sheet sizes, this option overrides 'page_size' variable. <*page_width*> and <*page_height*> must be defined in printer points (pt unit not needed in this variable definition). It is mandatory to provide both. See table C.7 for a reference on page size values.

page_orientation=<*Landscape*> | <*Portrait*> **Default= Landscape**

You can visualize your plots on 'Landscape' —sheet larger side from left to right— or 'Portrait' —sheet larger side from top to bottom—.

margin_left=<*#unit*> **Default= 1cm**

margin_right=<*#unit*> **Default= 1cm**

margin_upper=<*#unit*> **Default= 1cm**

margin_lower=<*#unit*> **Default= 1cm**

You can set page margins with those four variables. '#' means any real number. <*unit*> can be 'cm', 'in' or 'pt', **note** that there is no blank space between numbers and units.

foreground_color=<*color*> **Default= FGcolor**

background_color=<*color*> **Default= BGcolor**

'FGcolor', 'BGcolor', or any of the color names defined on table C.6. 'FGcolor' —default is 'black'— and 'BGcolor' —default is 'white'— are special color names that have a color defined by default, but once you have set one color for **foreground_color/background_color** custom-file variables you can use those special color names to assign background/foreground color to any other feature.

page_number=<*#*> **Default= 1**

'#' means any positive integer value greater or equal than 1. By default **`gff2ps`** fits all sequence nucleotides on the same page, but you can split that length in many pages as you need (useful when you have a long sequence).

zoom=<*first_nucleotide*>..<*last_nucleotide*> **Default= ∗..∗**

If you are just interested in visualizing a certain region on the whole sequence. You have four possible combinations to define values for this variable: '∗..∗', '<first_nucleotide>..∗', '∗..<last_nucleotide>', '<first_nucleotide>..<last_nucleotide>'. '∗' means to read nucleotide position from input files. 'first_nucleotide' and 'last_nucleotide' are nucleotide positions.

blocks_x_page=<*#*> **Default= 1**

Any positive integer value greater or equal than 1. If 0 is given then multiple vertical pages are generated, with a fixed track width. With this variable you can define how many plot areas you need to display by page. Nucleotides shown per page are divided into blocks number, and that is the nucleotide length for each block.

nucleotides_x_line=<*nucleotide_number*> **Default= 0**

Default value '0' makes program to read sequence nucleotide length from input data. Any integer value means to force those nucleotide number shown per page. Page number and blocks per page are re-calculated, also when you provide them as a variable or as an option, if you pass a nucleotides per line value, to the block and page number needed to view given nucleotides per lines value within a block. When leaving default value to 'nucleotides_x_line' (set as '0'), nucleotides needed to fill block width are calculated with following formula:

$$\text{nucleotides\_x\_line} = \frac{\text{sequence\_lenght}}{\text{page\_number} \times \text{blocks\_x\_page}}$$

where 'sequence_lenght' is the nucleotide sequence length read from input-files or the nucleotide difference for zoom start and end positions.

block_style=<*default*> | <*boxed*> **Default= default**

Blocks can be shown without frame —'default'— or with a frame around them —'boxed'—.

default_block_spacing_width=<*#unit*> **Default= 0.25cm**

'#' means any real number. <*unit*> can be 'cm', 'in' or 'pt'. Distance between one block to the next, also from the header area to the first block.

show_blocks_top-bottom=<*BOOLEAN*> **Default= 1**

'1' means show blocks Top-to-Bottom, '0' means Left-to-Right. When splitting output on several pages and having more than one block per page, you can start the next bottom block or the next right block —on the next page— with the last nucleotide of current block, so in second case you can join multiple pages with multiple blocks and nucleotides displayed on all the blocks are correlative from one page to the next.

header_style=<*none*> | <*default*> | <*boxed*> **Default= default**

Header can be defined as the 'title' region and its elements are title, sub-title, page-number, date and time. 'none' —does not plot header area and room left is used in plot area to enlarge blocks—, 'default' —no framed header—, 'boxed' —put a frame around header area—.

title=<*none*> | <*default*> | <*free-text_string*> **Default= default**

'none' does not print title, 'default' prints first input filename, and 'free-text_string' can be a title string given by user.

subtitle=<*none*> | <*default*> | <*free-text_string*> **Default= default**

'none' does not print sub-title, 'default' as 'none', 'free-text_string' can be a sub-title string given by user.

show_page_numbers=<*BOOLEAN*> **Default= on**

Boolean switch for displaying or not page numbering on top right corner of header area.

show_date=<*BOOLEAN*> **Default= on**

Boolean switch for displaying or not system date on right side of header area.

show_time=<*BOOLEAN*> **Default= on**

Boolean switch for displaying or not system time also on right side of header area.

major_tickmarks_num=<*#*> **Default= 10**

Any positive integer number. It defines number of major tickmarks shown per line, dividing nucleotides per line by the given parameter.

major_tickmarks_nucleotides=<*nucleotide_number*> **Default= -1**

Same as above but nucleotide distance between two neighbor major tickmarks is given, '-1' force program to use 'major_tickmarks_num' on 'nucleotides_x_line'.

minor_tickmarks_num=<*#*> **Default= 10**

Any positive integer number. It defines number of minor tickmarks shown between two major tickmarks, dividing available gap size between major tickmark by the given parameter.

minor_tickmarks_nucleotides=<*nucleotide_number*> **Default= -1**

Nucleotide distance between two neighbor minor tickmarks, '-1' force program to use 'minor_tickmarks_num' on 'nucleotides_x_line'.

show_grid=<*BOOLEAN*> **Default= on**

You can display vertical dotted lines for each minor tickmark with this boolean switch.

show_inner_scale=<*both*> | <*default*> | <*none*> | <*top*> | <*bottom*> **Default= both**

That inner scale is shown only if there are more than one strand displayed on the plot, and is used to delimitate strand areas (when three strands are visualized you get two inner rulers). ‘both’ and ‘default’ enable both nucleotide rulers, ‘none’ switch off both rulers, ‘top’ to show only the top ruler, and ‘bottom’ for showing only the bottom one.

show_outer_scale=<*both*> | <*default*> | <*none*> | <*top*> | <*bottom*> **Default= both**

There are two outer rulers delimitating block upper and lower sides. ‘both’ and ‘default’ enable both nucleotide rulers, ‘none’ switch off both rulers, ‘top’ to show only the top ruler, and ‘bottom’ for showing only the bottom one.

default_scale_width=<*#unit*> **Default= 0.25cm**

You can modify nucleotides ruler size, having a thinner or thicker rule, with smaller or larger font for nucleotide positions on major tickmarks. You can use ‘#cm’, ‘#in’, ‘#pt’. ‘#’ means any real number.

default_scale_spacing_width=<*factor*> **Default= 1**

You can increase or reduce the amount of space between each scale ruler and the source tracks. This space is obtained multiplying ‘factor’ by ‘default_scale_width’; so if you do not want free space between rulers and source tracks, you can set factor value to ‘0’.

nucleotide_scale=<*SCALE*> **Default= default**

<*SCALE*> ≡ <default> | <b> | <bases> | <k> | <kb> | <kilobases> | <m> | <mb> | <megabases>

This variable sets the scale in which the numbers on the nucleotide ruler should be displayed. ‘default’, ‘b’ and ‘bases’ show the nucleotide positions as the whole number (0, 100, 1000, 1000000); ‘k’, ‘kb’ and ‘kilobases’ show that in ‘Kb’ (0Kb, 0.1Kb, 1Kb, 1000Kb); ‘m’, ‘mb’ and ‘megabases’ show that in ‘Mb’ (0Mb, 0.0001Mb, 0.1Mb, 1Mb). Default in base numbers is useful for small sequences; on the other side, showing in megabases is useful for large sequences and for posters.

strand_show_forward=<*BOOLEAN*> **Default= on**
strand_show_reverse=<*BOOLEAN*> **Default= on**
strand_show_independent=<*BOOLEAN*> **Default= on**

Boolean switches for displaying strand areas: ‘on’ enables to plot area and ‘off’ disables. You may be not interested in visualize forward strand area —because it is empty or you want to enlarge reverse area to focus on some details— so you can disable this area. At least one strand area must be shown, this allows you to display one, two or three strands (including all features without a defined strand ‘.’ in the central area). Strand order for strand areas within the block is always from top to bottom: forward elements, no-strand elements and reverse elements (upper, middle and lower block areas respectively).

show_left_source_label=<*BOOLEAN*> **Default= true**

Boolean switch for printing source-labels on left side of blocks: ‘true’ enables and ‘false’ disables.

left_source_label_width=<*#unit*> **Default= 2cm**

Left source-labels width. You can use ‘#cm’, ‘#in’, ‘#pt’. ‘#’ means any real number.

show_right_source_label=<*BOOLEAN*> **Default= false**

Boolean switch, shows source-labels on right side of blocks: ‘true’ enables, ‘false’ disables.

NOTE: You can have source-labels on both block sides, left and right, but it takes some precious drawing area that will be available for visualizing your sequence. Also you can reduce label width, providing more room for that drawing area.

right_source_label_width=<*#unit*> **Default= 2cm**

Right source-labels width: You can use ‘#cm’, ‘#in’, ‘#pt’. ‘#’ means any real number.

sort_tracks_by_sequence=<*BOOLEAN*> **Default= on**

By default, when multiple sources with multiple sequence IDs are going to be plotted on the same page tracks are sorted by sequence (putting closer all sources for the same sequence). You can sort them by source (placing together all sequences for the same source) too.

default_track_width=<*#unit*> **Default= 1cm**

Default source tracks ‘vertical’ width: You can use ‘#cm’, ‘#in’ and ‘#pt’, where ‘#’ means any real number. You can enlarge or reduce track width for all sources on plot here.

default_track_spacing_width=<*#unit*> **Default= 0.25cm**

Default source tracks spacer ‘vertical’ width: You can use ‘#cm’, ‘#in’ and ‘#pt’, where ‘#’ means any real number.

group_label_scale=<*factor*> **Default= 1**

You can increase or reduce the size of group labels (globally) by ‘factor’.

position_label_scale=<*factor*> **Default= 1**

You can increase or reduce the size of coords labels (nucleotide start-end positions) by ‘factor’.

frame_unknown_color=<*COLOR*> **Default= orange**
frame0_color=<*COLOR*> **Default= blue**
frame1_color=<*COLOR*> **Default= red**
frame2_color=<*COLOR*> **Default= green**

<*COLOR*> ≡ <color> | <default> | <FGcolor> | <BGcolor>

‘color’ can be any of the color names defined on table C.6. ‘default’ is black, ‘FGcolor’ uses the color you have defined for ‘foreground_color’ and ‘FGcolor’ uses the color set for the ‘background_color’ (see section 4.1.1). Those four variables are global definition for frame-remainder fill-shape mode, and are used when you define for any of the ‘GFF-feature_key’ in the feature block of your custom file (see this variable definition on page 24):

<*GFF-feature_key*>**::fill_shape_mode=**<*frame-remainder*>

You can find how **gff2ps** calculates remainder for a given frame in page 9.

show_positions=<*BOOLEAN*> **Default= false**

‘true’ or ‘false’. Shows nucleotide start-end positions —in forward coordinates— for all elements on the plot.

min_group_separation=<*nucleotide_number*> **Default= 10**

Number of nucleotides from 0 to n defining the minimum nucleotide distance between two consecutive groups to avoid overlapping on same track when overlapping groups unfolding into several tracks is enabled (see variable definition for 'unfold_grouped_ungrouped', 'unfold_grouped_line' and 'unfold_ungrouped_line' in section 4.1.4, page 26).

4.1.2 GFF-element Features

You must define a <*GFF-feature_key*> —from third field on GFF-file— to which is going to assign the variable value, it can be a regular expression —see section 4.2.2—, an standard key value —see appendix A— or a GFF-feature that you have already defined in your GFF-file. You already know that this feature-key must not be case-sensitive, and **`gff2ps`** always convert them to lower case characters.

<*GFF-feature_key*>**::feature_color=**<*COLOR*> **Default= default**

<*GFF-feature_key*>**::feature_stroke_color=**<*COLOR*> **Default= default**

<*COLOR*> ≡ <color[..color[..]color]> | <default> | <FGcolor> | <BGcolor>

'feature_color' defines fill color for GFF-features shapes and 'feature_stroke_color' the color for border line of those shapes. Color names are from table C.6. 'default' is black, 'FGcolor' uses the color you have defined for 'foreground_color' and 'FGcolor' uses the color set for the 'background_color' (see section 4.1.1). For some values on 'fill_shape_mode' and 'fill_vector_mode' —**1_color**, **2_color**, **3_color**—, you can set up three colors using two consecutive dots between each color '**..**' without blank spaces, but if you choose one of those fill modes and assign less colors than they need then, default values are given by **`gff2ps`**.

<*GFF-feature_key*>**::shape=**<*shape_name*> **Default= box**

Defining feature drawing shape. See table C.1 for available values, default is defined as 'box'.

<*GFF-feature_key*>**::fill_shape_mode=**<*fill_mode*> **Default= default**

Setting the way to fill the shapes for the given GFF-feature. See table C.4 for available values.

<*GFF-feature_key*>**::fill_vector_mode=**<*fill_mode*> **Default= default**

Vectors are special features that allows to display position-score data-sets. Here you can choose how you want to visualize those vectors in a gradient colored line. See table C.5 for available values.

<*GFF-feature_key*>**::vert_align=**<*align_mode*> **Default= default**

Vertical alignment for features is referred to source baseline. Available values are only 'default' —centered on source track baseline, half shapes are drawn over it—, 'center' —same as previous— and 'mirror' —half shapes are drawn under the track baseline— (see table C.2).

<*GFF-feature_key*>**::layer=**<*#*> **Default= 0**

First is made a sorting for feature going to be placed on lines on this variable. By default all features are drawn on lower layer (that is '0'), but you can move them to upper layers by setting a greater number for '#' than default. Within a layer, larger elements are displayed below the shorter ones. When groups are shown as GFF-features they are always sent to lowest layer ('0').

<*GFF-feature_key*>**::label=**<*LABEL*> **Default= default**

<*LABEL*> ≡ <++none++> | <++default++> | <user-def_string>

Printing labels for GFF-elements (still not implemented). ‘none’ means not to display element label, ‘default’ shows GFF-feature value, and a ‘user-def_string’ represents free-text user defined label that is printed in place of GFF-feature value.

<*GFF-feature_key*>**::show_feature=**<*BOOLEAN*> **Default= on**

Boolean switch to visualize or not any GFF-feature: ‘on’ shows feature, ‘off’ does not show feature.

<*GFF-feature_key*>**::show_feature_positions=**<*BOOLEAN*> **Default= off**

Boolean switch to visualize or not start and end nucleotide positions for one GFF-feature: ‘on’ shows positions, ‘off’ does not show positions.

4.1.3 Group Features

A <*GFF-group_key*> —from field nine on GFF-file— must be defined as a regular expression —see section 4.2.2— which include a group value that you have defined in your GFF-file. The reason for such “constrain” is that **gff2ps** defines groups as a compound string containing sequence name, source name and group name when the group is defined in the GFF file, to avoid mix groups defined with same string from different sequences or sources. If there is no group in ninth field, program generates an internal group tag for each ungrouped feature record, that construct is made by joining record line number, sequence name —first field—, source field —the third—, and strand —on seventh field—; so when trying to change variables for those groups you may need to use regular expressions. Also you can force the program to re-write default custom file, those groups will be printed in that file (see section 4.2 for further details on custom files generation). You can visualize groups not only as a grouping bracket or line, also as a GFF-feature, by giving a ‘group_shape’ (default value for this variable is ‘none’).

<*GFF-group_key*>**::feature_color=**<*COLOR*> **Default= default**

<*GFF-group_key*>**::feature_stroke_color=**<*COLOR*> **Default= default**

To force same fill-color for all GFF-features within that group, overriding any ‘feature_color’ definition for its elements from the GFF-features custom-file block. See ‘COLOR’ definition in next variable.

<*GFF-group_key*>**::group_color=**<*COLOR*> **Default= default**

<*COLOR*> ≡ <color[..color[..]color]> | <default> | <FGcolor> | <BGcolor>

‘feature_color’ defines fill color for all GFF-features shapes and ‘feature_stroke_color’ the color for border line of those shapes, that belong to the same group. Also if you like to print the group as an extra GFF-feature you can set fill-color here. Color names are from table C.6. ‘default’ is black, ‘FGcolor’ uses the color you have defined for ‘foreground_color’ and ‘FGcolor’ uses the color set for the ‘background_color’ (see section 4.1.1). For some values on ‘fill_shape_mode’ and ‘fill_vector_mode’ —‘1_color’, ‘2_color’, ‘3_color’—, you can set up to three colors using two consecutive dots between each color ‘**..**’ without blank spaces, but if you choose one of those fill modes and assign less colors than they need then default values are given by **gff2ps**.

<*GFF-group_key*>**::group_shape=**<*shape_name*> | <*default*> **Default= none**

Defining group drawing shape, visualizing the group as an extra GFF-feature. See table C.1 for available values, default is defined as 'none'.

<*GFF-group_key*>**::fill_shape_mode=**<*fill_mode*> **Default= default**

Setting the way to fill the shapes for the given GFF-feature. See table C.4 for available values.

<*GFF-group_key*>**::fill_vector_mode=**<*fill_mode*> **Default= default**

Vectors are special features that allow to display position-score data-sets. Here you can choose how you want to visualize them in a gradient colored line. See table C.5 for available values.

<*GFF-group_key*>**::group_line=**<*line_type*> **Default= default**

You can select which kind of line you would like to 'underline' a group from start to end. See table C.3 for available values.

<*GFF-group_key*>**::group_line_color=**<*COLOR*> **Default= default**

<*COLOR*> ≡ <color> | <default> | <FGcolor> | <BGcolor>

'default' is black, 'FGcolor' uses the color you have defined for 'foreground_color' and 'FGcolor' uses the color set for the 'background_color' (see section 4.1.1). 'color' may be any of the color names defined on table C.6.

<*GFF-group_key*>**::vert_align=**<*align_mode*> **Default= default**

Vertical alignment for groups is referred to source baseline, overrides any definition for all features contained in that group. Available values are only 'default' —centered on source track baseline, half shapes are drawn over it—, 'center' —same as previous— and 'mirror' —half shapes are drawn under the track baseline— (see table C.2).

<*GFF-group_key*>**::label=**<*LABEL*> **Default= default**

<*LABEL*> ≡ <++none++> | <++default++> | <user-def_string>

Printing labels for GFF-groups. 'none' means not to display element label, 'default' shows group tag value from input files, and a 'user-def_string' represents free-text user defined label that is printed in place of group tag value.

<*GFF-group_key*>**::show_group=**<*BOOLEAN*> **Default= on**

Boolean switch for visualizing any group (and also all GFF-elements which it will contain): 'on' shows group, 'off' does not show group.

<*GFF-group_key*>**::show_group_positions=**<*BOOLEAN*> **Default= off**

Boolean switch to visualize or not start and end nucleotide positions for all GFF-features for that group: 'on' shows positions, 'off' does not show positions.

4.1.4 Source Features

We obtain <*GFF-source_key*> from GFF-records second field, it can be defined as regular expression —see section 4.2.2— or a source that you have defined in your GFF-file.

<*GFF-source_key*>**::feature_color=**<*COLOR*> **Default= default**

<*GFF-source_key*>**::feature_stroke_color=**<*COLOR*> **Default= default**

‘feature_color’ defines fill color for all GFF-features shapes and ‘feature_stroke_color’ the color for border line of those shapes, that belong to the same source. Force same fill-color for all GFF-features within that source tracks, overriding any ‘feature_color’ definition for its elements from the GFF-features custom-file block.

<*GFF-source_key*>**::group_color=**<*COLOR*> **Default= default**

When printing groups as an extra GFF-feature you can override ‘group_color’ definition for all groups on that source from the group custom-file block.

<*COLOR*> ≡ <color[..color[..]color]> | <default> | <FGcolor> | <BGcolor>

Color names are from table C.6. ‘default’ is black, ‘FGcolor’ uses the color you have defined for ‘foreground_color’ and ‘FGcolor’ uses the color set for the ‘background_color’ (see section 4.1.1). For some values on ‘fill_shape_mode’ and ‘fill_vector_mode’ —‘1_color’, ‘2_color’, ‘3_color’—, you can set up to three colors using two consecutive dots between each color ‘..’ without blank spaces, but if you choose one of those fill modes and assign less colors than they need then default values are given by **`gff2ps`**.

<*GFF-source_key*>**::left_label=**<*LABEL*> **Default= default**

<*GFF-source_key*>**::right_label=**<*LABEL*> **Default= default**

<*LABEL*> ≡ <++none++> | <++default++> | <++sequence++> | <++source++> | <++both++> | <++info++> | <user-def_string>

Printing labels for sources: ‘++none++’ means not to display element label, ‘++default++’ shows source or sequence string value from input files (depends on ‘sort_tracks_by_sequence’ variable), ‘++sequence++’ force to show sequence name for those source tracks, ‘++source++’ does the same as previous but showing source name, ‘++both++’ shows sequence and source names, ‘++info++’ prints the same as ‘++both++’ plus two extra codes (first for ‘G’rouped-‘U’ngrouped-‘M’ixed tracks and second for ‘+’/‘-’ strand), and a ‘user-def_string’ represents free-text user defined label that is printed in place of source tag value.

<*GFF-source_key*>**::show_left_source_label=**<*BOOLEAN*> **Default= on**

<*GFF-source_key*>**::show_right_source_label=**<*BOOLEAN*> **Default= off**

By default only appears left track labels on the plot but you can visualize both, left and right, or only right, playing with these two variables.

<*GFF-source_key*>**::source_label_scale=**<*factor*> **Default= 1**

You can increase or reduce the size of source label by ‘factor’. This is useful to resize those labels when modifying ‘track_scale’ factor.

<*GFF-source_key*>**::source_style=**<*default*> | <*boxed*> **Default= default**

‘default’ —no framed source-track— or ‘boxed’ —put a frame around source-track—.

<*GFF-source_key*>**::source_line=**<*line_type*> **Default= default**

Setting line-style for source baseline. See upper panel on table C.3 for available values.

<*GFF-source_key*>**::source_line_color=**<*COLOR*> **Default= red**

Defining color for source baseline.

<*COLOR*> ≡ <color> | <FGcolor> | <BGcolor>

‘FGcolor’ uses the color you have defined for ‘foreground_color’ and ‘FGcolor’ uses the color set for the ‘background_color’ (see section 4.1.1). ‘color’ can be any of the color names defined on table C.6.

<*GFF-source_key*>**::vert_align=**<*align_mode*> **Default= default**

Vertical alignment for source baselines within the source tracks. See table C.2 for all available values.

<*GFF-source_key*>**::range=**<*none*> | <*default*> | <*#..#*> **Default= default**

Defining range of lower-upper source-scores: with ‘none’ all scores are set to ‘1’, ‘default’ takes score range obtained from input files, and ‘#..#’ forces range to defined values —it works like ‘zoom’ layout variable: ‘#..#’ is like ‘default’, ‘min..#’ takes max from input file, ‘#..max’ takes min from input file, and ‘min..max’ defines both, upper and lower score limits—.

<*GFF-source_key*>**::track_scale=**<*#*> **Default= 1**

Any positive real number. This variable allows you to resize one or more source-tracks, giving a different aspect ratio than the others (i.e. twice wider or half wider than normal tracks).

<*GFF-source_key*>**::track_spacing_scale=**<*#*> **Default= 0.25**

Any positive real number, this variable allows you to resize the amount of space between two tracks of different consecutive sources (like an spacer for source-tracks).

<*GFF-source_key*>**::keep_feature_label_space=**<*BOOLEAN*> **Default= on**

Boolean switch that controls how much of the source track width is assigned to draw features and how much to feature labels. ‘on’ reserves half of the track width to labels and the other half for the features, ‘off’ forces the features to be drawn fitting the full source track width.

<*GFF-source_key*>**::unfold_grouped_ungrouped=**<*BOOLEAN*> **Default= off**

Boolean switch to split grouped elements from ungrouped: ‘on’ grouped and ungrouped elements shown in two tracks, ‘off’ all grouped and ungrouped elements fitting only one track.

<*GFF-source_key*>**::unfold_grouped_line=**<*BOOLEAN*> **Default= on**

Boolean switch to split overlapping grouped elements in several tracks: ‘on’ split groups, ‘off’ all grouped elements fitting only one track.

<*GFF-source_key*>**::unfold_ungrouped_line=**<*BOOLEAN*> **Default= on**

Boolean switch to split overlapping ungrouped elements in several tracks: ‘on’ split ungrouped elements, ‘off’ all ungrouped elements to one track.

<*GFF-group_key*>**::show_source_positions=**<*BOOLEAN*> **Default= off**

Boolean switch to visualize, or not, start and end nucleotide positions for all GFF-features for that source: ‘on’ shows positions, ‘off’ does not show positions.

4.1.5 Solving conflicts among variables

Here we should try to give you an idea of how **`gff2ps`** solves conflicts among variables. You must know that the program now warns the user if he provided a wrong variable name in any section, layout, features, groups or sources. If you define a wrong value for any of the variables, then the program assumes the default value for that variable but is not warning on that.

In those cases in which one variable can be defined in more than one section the program uses for the plot the value for the higher level definition. As example, the variable '`feature_color`' that can be defined in the feature, the group and the source sections. If you have defined the variable in the feature and in the group sections, the value for all the features within the group for which you have defined the variable is set from the group section, all the other features not in that group are set by the variable on the feature section.

The program has all the variables set on default values; if default custom file exist, those variables redefined in that file override the defaults. If you provide an extra custom file from command-line, its variable definitions will modify the corresponding variable values. Last changes are given by some command-line options, that reset some of the layout variables.

There is one special case in which one variable definition excludes another one depending on the input data. That happens for the '`fill_shape_mode`' and '`fill_vector_mode`' variables, the first defines the way in which are going to be filled those features that are related to a shape (almost all GFF records), the second sets the color filling procedure for those features that represent a scoring vector (those special records in which the group tag is "Vector", and that provide a set of scores, see page 9). Depending on the input record (if it is a scoring vector or not) **gff2ps** will use one variable or the other to define the procedure that the user would like for color filling.

4.2 Custom Files

gff2ps can handle two customizing files in which you can define new values for the variables used on plots:

- Default Custom File
- Extra Custom File

By default there is no custom file, you must force the program to write Default Custom File, but program works as well without them because it has a default values array coded within. Three command-line options can help you with custom files:

-d Writes (or re-writes if exist) Default Custom File. Default filename for this custom file is '`.gff2psrc`', but you can define another Default Custom filename with the environment variable "GFF2PS_CUSTOMFILE" —further information on **gff2ps** environment variables on section 3.1—.

-D <*default_custom_filename*> If you want to create a new Default Custom-file, with the given filename, and without overwriting '`.gff2psrc`', you must use this option. Although you have to set the environment variable "GFF2PS_CUSTOMFILE" as the new custom-file name if you want **gff2ps** working with it. An example will be:

1. Make **gff2ps** to generate `newdefs` Custom File:
   [*cshell*]$ `gff2ps -D newdefs -- file1.gff file2.gff > output1.ps`
2. Then define the new Default Custom File name:
   - Using a Bourne-Shell (e.g. bash):
     [*bshell*]$ `export GFF2PS_CUSTOMFILE="newdefs"`

- Using a C-Shell:
  [*cshell*]$ `setenv GFF2PS_CUSTOMFILE "newdefs"`

3. Now you can work with this Default Custom File on other inputs, without passing custom file option to **gff2ps**:
   [*cshell*]$ `gff2ps file3.gff file4.gff > output2.ps`

-C <*custom_filename*> Loads given Custom File and appends it to Default Custom File or to program defaults if there is no Default Custom File defined.

You may guess why we have defined two different Custom Files and what you can do with them. The answer is that you can modify Default Custom File the way you like and use it as a global defaults for your plots, then you can modify many variables as you only need for one plot with "-C" option and the secondary Custom File, which has not to be larger as Default Custom File. You can have as many Custom Files as you want, and use each of them in several plots or in the same one, but you only can append one Custom File to Default Custom File.

```
[cshell]$ cat > plotAdefs
# F # (Return)
exon::shape=arrow (Return and CTRL+C)
[cshell]$ gff2ps -C plotAdefs -- file1.gff file2.gff > output1b.ps
```

Figure 4.1: Writing our first custom file. That one only contains one variable definition for GFF-elements, in this case we are modifying 'shape' value for 'exon' GFF-feature to 'arrow' in `output1b.ps` plot.

Following the previous example on "-D" option, we can create a small Custom File and pass it to the program which adds to `"newdefs"` variable definitions. Figure 4.1 provides an example for a simple and very short extra custom file. Last command-line implies to modify 'shape' variable for 'exon' GFF-feature from `"newdefs"` to `"plotAdefs` value for this variable. When writing Custom File, you can take a look on variable definition from Default Custom File, copying and pasting lines defining a variable from it.

You can use '#' for comments. Those lines starting with '#' are understood as comment lines. You can place a comment at the end of a variable definition, but there must be almost one blank space between variable definition and '#'. Empty lines, comment lines and lines that does not apply on variable definition format (see 4.1) are ignored when reading Custom Files. There is only one exception, the block separator comment that is needed to define where each of the four blocks starts in which you can divide Custom Files.

4.2.1 The Block Separator Comment

Block Separator Comments are fixed string comments that **gff2ps** treat as headers for delimiting each of the four blocks of variables you can define into a Custom File. We have defined those four blocks at the beginning of this section: Layout, GFF-elements, groups and sources. When writing the Default Custom File, **gff2ps** always creates the four blocks, but you must specify at least one for your Custom File as we could see in the above example of working with user-defined custom files (figure 4.1). In that example we have set the variable 'shape' for 'exon' as an 'arrow', because it is a GFF-feature variable definition we wrote the Block Separator '# F #' to tell the program we were re-defining a variable from GFF-elements Block.

```
#
# Optional Header
#              (gff2ps generates a standard header when creates Default Custom File)
#
                                              (Comments and empty lines are skipped)
# L #                                         (This is Block Separator for Layout)
<variable_name1>=<value>␣# blah,blah,blah...  (Note that there is a blank space
                                               —shown here as '␣'— before #)
   ⋮        ⋮        ⋮
# blah, blah, blah...              (Extra comment-lines can be added where you need)
   ⋮        ⋮        ⋮
<variable_namen>=<value>
# F # blah, blah, blah...  (You can place extra comments also here, after second #)
<GFF-feature_key1>::<feature_variable_name>=<value>
         ⋮                 ⋮                 ⋮
<GFF-feature_keyn>::<feature_variable_name>=<value>
# G #
<GFF-group_key1>::<group_variable_name>=<value>
         ⋮                 ⋮                 ⋮
<GFF-group_keyn>::<group_variable_name>=<value>
# S #
<GFF-source_key1>::<source_variable_name>=<value>
         ⋮                 ⋮                 ⋮
<GFF-source_keyn>::<source_variable_name>=<value>
```

Figure 4.2: Custom Files general structure, pointing on block separator comments usage for delimiting variable definition into four groups: layout, GFF-elements, GFF-groups and GFF-sources.

The blocks corresponding separators are:'# L #' for Layout,'# F #' for GFF-elements, '# G #' for Groups, and '# S #' for Sources. You can write extra text before the second '#' and use those headers also as comments, but you must remember to maintain the first five characters as we have defined. Figure 4.2 shows general structure for Custom Files and block separator comments usage. Block ordering is not relevant but variables defined on each block can not be placed on the others.

4.2.2 Using Regular Expressions in Custom Files

Main **gff2ps** scripting code is written in GNU awk, you may refer to awk reference manuals[1,2] to know more about how this language manages with regular expressions, but we should try to summarize the main features. A Regular Expression is a notation used to specify and match strings. You can get more info about regular expressions in Friedl's book[3]. Table 4.1 shows the main players on regular expressions and how you can combine them into higher constructs for parsing your feature space.

When working on GFF-files containing a large number of GFF-features, Groups or Sources,

[1] Aho, A.; Kernighan, B.W.; Weinberger, P.J. "The AWK Programming Language" Addison-Wesley Publishing Co.; first edition (1988).

[2] Dougherty, D.; Robbins, A. "sed & awk" Ed. O'reilly & Associates, Inc; Second edition (1997).

[3] Friedl, J.E.F."Mastering Regular Expressions" Ed. O'reilly & Associates, Inc; first edition (1997).

- Regular expression metacharacters are:
  `\ ^ $ . [ ] | ( ) * + ?`
- Basic regular expression is one of the following:

  a nonmetacharacter, such A, that matches itself.

  an escape sequence that matches a special symbol (\t matches a tab).

  a quoted metacharacter, as *, that matches the metacharacter literally.

  ^ matches the beginning of a string.

  $ matches the end of a string.

  . matches any single character.

  a character class, [ABC] matches any of the characters A, B, or C. Character classes may include abbreviations, [a-z] matches only lower case letters.

  a complemented character class, [^0-9] matches any character except a digit.
- Operators that combine regular expressions into larger ones:

  alternation, a|b matches A or B.

  concatenation, AB matches A immediately followed by B.

  closure, A* matches zero or more A's.

  positive closure, A+ matches one or more A's.

  zero or none, A? matches null string or A.

  parentheses, (r) matches same string as r does.

Table 4.1: Describing Regular Expressions (excerpt from note 1).

regular expressions allow you not to spend your time defining variables for each one (either you will be interested on displaying each one in a different way). You can be interested in changing one variable value for few, many or all elements and this can be done faster and easily with regular expressions. You can not use them on the Layout Block, because variables defined within this block are not linked to any feature; in the three other blocks you must define a feature from GFF-files that is going to be drawn as we are setting its variables values. Then we can apply regular expressions when assigning one value to a variable for multiple features at once, an example of string pattern matching using regular expressions applied to **gff2ps** is on table 4.2.

How can we define regular expressions on **gff2ps** ? We must quote regular expression between two slashes ('/') in <*plot_element*> field of variable definition on custom file, as it is described on the following pattern:

/regular_expression/::<*variable_name*>**=**<*value*>

Where 'regular_expression' could be any regular expression construct in compliance with table 4.1 rules. The only exception to this general definition is made for the 'any string' matching expression, with which you can select all plot elements, that can be written as '*' in place of '/.*/', so you can place on your custom file a line like this:

***::**<*variable_name*>**=**<*value*>

Defining a words list:

$$cds,\ codon,\ est,\ 5'est,\ 3'est,\ splicesite,\ 5'splicesite,\ 3'splicesite,$$
$$start,\ stop\_codon,\ utr,\ 5'utr,\ 3'utr,\ group.3,\ file.gff$$

The following table shows matching results for left regular expressions over previous words list:

Expression	**Matches**	**Examples**
c	the nonmetacharacter c	cds, $codon$, $splicesite$, $5'splicesite$, $3'splicesite$, $stop\_codon$
\\.	escape sequence or literal character .	$group.3$, $file.gff$
ˆc	c at the beginning of string	cds, $codon$
utr\$	utr at the end of string	utr, $5'utr$, $3'utr$
.	any character	all the words list
ˆ...\$	any string containing exactly three characters	cds, est, utr
ˆ(5′)?u	any string starting with $5'u$ or u	utr, $5'utr$
ˆ5′[ues]	any string starting with $5'$ followed by u or e or s	$5'est$, $5'splicesite$, $5'utr$

Table 4.2: Working with Regular Expressions, some examples.

that is equivalent to:

/.∗/::<*variable_name*>**=**<*value*>

You may be interested also in changing any attributes for those features that do not match your regular expressions. **gff2ps** can deal with that, but it's only allowed with the regular expression variable definition general format. You can deny the regular expression using '**!**' (exclamation mark) as it's drafted here:

!/regular_expression/::<*variable_name*>**=**<*value*>

so, you can modify the variable for all the features that are not accomplishing such regular expression. Figure 4.3 shows many examples on how to apply regular expressions in **gff2ps** custom files.

```
# F #
exon::shape=arrow                 # exon shape as arrow.
/^5.*/::shape=box                 # all features starting with '5' are shown with a box.
*::feature_color=lightred         # fill color for 'ALL' features is now 'lightred'.
# G #
/.(QT)?s$/::label=QTgp            # define 'QTgp' as label for any group ending on '.QTs' or '.s'.
/CLONE[AB].*/::show_group=off     # Any group matching 'CLONEA' or 'CLONEB' is not shown.
# S #
!/.*blast.*/::vert_align=top      # 'ALL' sources NOT matching '.*blast.*' vertical align
                                  # for baseline are set to top.
```

Figure 4.3: Using regular expressions on **gff2ps** custom files

4.3 Command Line Options

```
gff2ps -v -n -s <param> GFF_file
gff2ps -v -n -s <param> -- GFF_file
gff2ps -vn -s <param> GFF_file
gff2ps -vn -s <param> GFF_file
gff2ps -s <param> -n -v GFF_file
```

Figure 4.4: Reading options from command-line, all the above command lines are equivalent. 'v' and 'n' are options without parameter, 's' needs a parameter.

All options must be passed before GFF input-files. Option string is read from left to right and, as in Unix, you can split together options that does not need parameters, because **gff2ps** checks options with 'getopts' —see figure 4.4—. Note that there is a blank space between option and parameter.

A double hyphen ('--') notifies shell-script 'getopts' function that there are no options left —second line on figure 4.4—. Although it is optional, we recommend its use for preserving clarity and showing you where option definition finishes and input files starts. Also, if you switch on any command-line option that needs a parameter, with double hyphen you can detect that you have forgotten its parameter instead of getting the first filename as parameter; such error is more difficult to parse because you are providing that filename but you do not get expected results for it, and that is caused by a wrong option definition, not due to input files.

A list of available command-line options can be found in table B. The main utility of command-line options is to modify some general layout variables for a plot without having to edit the Custom Files again, but there are seven options that are not represented by a Custom-File Variable: 'h' and 'H' were explained on section 3.2.2, page 14; 'v' and 'V' were defined on section 3.2.1, page 13; 'd','D' and 'C', that were described on section 4.2, page 27. Here, in this section we are going to show the Custom File Variable equivalents enclosed within a brackets for the rest of Command-line Options, so you can have an idea of what can be modified easily for one specific plot. Remember that Layout Variables are defined by defining a Value to a Variable Name —see 4.1—.

-a — **show_copyright_line**=**on**

This option disables the tiny copyright line at the bottom of your plots.

-s <*page_size*> — **page_size**=<*page_size*>

<*page_size*> is a page label from table C.7. By default is defined as 'a4'.

-p — **page_orientation**=**Portrait**

Switches page orientation to Portrait; if not passed, program first looks at custom files for 'page_orientation' definition and if that fails uses inner default value 'Landscape'.

-G <*color*> — **foreground_color**=<*color*>

You can change foreground color —text, outlines and tickmarck rules are drawn with that color—. <*color*> names are defined on table C.6, default value is 'FGcolor' (black).

-g <*color*> **background_color**=<*color*>

This sets background color for pages —defines fill color for paper sheet—. <*color*> names are defined on table C.6, default value is 'BGcolor' (white).

-P <*#*> **page_number**=<*#*>

<*#*> can be any integer value greater or equal than 1, which is the default value. Setting any other number forces program to split into given page number the input sequence length.

-S <*#*> **zoom=#..***

This option allows you to define starting nucleotide different to sequence first position (by default '0'). If '-E' is not given, then you get a zoom from <*#*> to last sequence position.

-E <*#*> **zoom=*..#**

Here you can define last nucleotide to be shown on plot (by default is last position from given sequence input-files). If '**-S**' is not given then you get a plot zoom from first sequence position to <*#*>.

NOTE: Giving both command-line options '-S <*#*>' and '-E <*#*>' you con obtain same result as custom-file variable 'zoom=#..#'.

-B <*#*> **blocks_x_page**=<*#*>

<*#*> can be any integer value greater or equal than 1, which is the default value. Providing this option you can visualize more than one block per page. Remember that each block contains three areas (forward, reverse and no-strand), within source tracks are displayed. This feature allows you to divide total nucleotides shown per page on '<*#*>' blocks —useful when working with large page formats (A3 to A0)—.

-N <*#*> **nucleotides_x_line**=<*#*>

<*#*> is the number of nucleotides you want to visualize within a block. Default is '0' that is to obtain sequence length from input-files, and use it to fit in the page by blocks space.

NOTE: Page number and blocks per page are re-calculated, also whether you provide them as a variable or as an option, if you pass a nucleotides per line value, to the block and page number needed to view given nucleotides per lines value within a block.

NOTE: See variable description for 'nucleotides_x_line' on page 18, to see how **`gff2ps`** calculates nucleotides per line when default value '0' is given.

-b **show_blocks_top-bottom**=2

If this option is enabled, then block offset is taken from left to right, increasing the nucleotides shown from left to right and, in case of multiple blocks per page, from top to bottom once we have reached last page. This option is useful when you want to splice all the pages in a mini-poster, and you have more than one block, because block nucleotides are shown correlative for all the blocks. If not given, last nucleotide position on previous block is, by default, the first on next block within a page.

-L **header_style**=none

Disables header area, so you do not have title, subtitle, page numbering, also date and time stamping, and room left can be exploited as drawing area for blocks.

-T <*"free-text_string"*> **title**=**free-text_string**

While in variable definition you do not need to quote <*free-text_string*>, you must do when passing this string in command-line due to shell requirements. String given is placed as plot title. Default value is the name of the GFF input-files.

-t <*"free-text_string"*> **subtitle**=**free-text_string**

Quotation is explained above. Given string is taken as a plot subtitle. By default there is no subtitle.

-l **show_page_numbers**=**off**

Disables page numbering that is shown by default in top right corner of header.

-O **show_date**=**off**

Disables date stamp shown by default in right margin of header.

-o **show_time**=**off**

Here you can disable time stamping in header right margin.

-M <*#*> **major_tickmarks_num**=<*#*>

Number of major tickmarks shown per line, it takes nucleotides per lines and divides it by the given parameter.

-K <*#*> **major_tickmarks_nucleotides**=<*#*>

Same as above, but passing a fixed nucleotides length between major tickmarks that is used when '-M' is defined as '-1' (default value).

-m <*#*> **minor_tickmarks_num**=<*#*>

Number of minor tickmarks shown per line, it takes nucleotides length between two major tickmarks and divides it by the given parameter.

-k <*#*> **minor_tickmarks_nucleotides**=<*#*>

You can force a fixed nucleotide length from minor tickmark to next one, using this option and leaving default value for '-m' (that is also '-1').

-w *or* **-f** **strand_show_forward**=**off**

Switch off plotting the forward strand block area for all the blocks and pages. Useful when you are interested only in reverse or no-strand features.

-c *or* **-r** **strand_show_reverse**=**off**

Switch off plotting of reverse strand block area for every block and page. Useful when you are only interested in forward or no-strand features.

NOTE: '-w' and '-c' are used as abbreviations of WATSON and CRICK, respectively, as is defined in some genomic analysis programs.

-i **strand_show_independent**=**off**

Switch off plotting for strand independent features area, because you only have features on forward and reverse, and/or you are interested to not have the no-strand area.

NOTE: At least one of the three strand related areas must be present in your plot. Any combination of two areas is allowed, and their ordering is always preserved —top area for forward-strand, middle area for no-strand and bottom area for reverse— even though you have enabled display for any pair of them or all three.

-0 <*color*> **frame0_color**=<*color*>

Changing color which fills the frame 0 halves of GFF-elements when variable 'fill_shape_mode' is set as 'frame-remainder'. Color names are available at table C.6.

-1 <*color*> **frame1_color**=<*color*>

Changing the fill color for frame 1 halves of a GFF-element when variable 'fill_shape_mode' is set as 'frame-remainder'. Color names are available at table C.6.

-2 <*color*> **frame2_color**=<*color*>

Fill color for frame 2 is redefined as given color name when variable 'fill_shape_mode' is set as 'frame-remainder'. Color names are available at table C.6.

-3 <*color*> **frame_unknown_color**=<*color*>

Same as above descriptions, but modifying color assigned to unknown frame GFF-elements (when frame field on GFF-records has a '.' or an invalid frame —different from '0', '1' or '2'—).

-n **show_positions**=**on**

Switch on showing start and end nucleotide coordinates for all displayed features on plot.

Appendix A : GFF Feature Table

In this section we want to summarize some of the DDBJ/EMBL/GenBank feature keys used in genomic analysis, and include also a brief description of their meaning. For further information see the DDBJ/EMBL/GenBank feature table definition at:

`http://www.ebi.ac.uk/embl/Documentation/FT_definitions/feature_table.html`

GENE FEATURES

gene : DNA region that spans the information for a protein (alternative splice forms are also covered).

prim_transcript : primary transcript (unprocessed RNA); includes 5' clipped region (5'clip), 5' untranslated region (5'UTR), coding sequences (CDS, exon), intervening sequences (intron), 3' untranslated region (3'UTR), and 3' clipped region (3'clip).

precursor_RNA : any RNA species that is not yet the mature RNA product; may include any of the above listed elements.

mRNA : messenger RNA; includes 5'untranslated region (5'UTR), coding sequences (CDS, exon) and 3'untranslated region (3'UTR).

tRNA : mature transfer RNA, mediates the translation of a nucleic acid sequence into an amino acid sequence.

rRNA : mature ribosomal RNA ; the RNA component of the ribosome which assembles amino acids into proteins.

snRNA : small nuclear RNA; any one of many small RNA species confined to the nucleus; several of the snRNAs are involved in splicing or other RNA processing reactions.

intron : a segment of DNA that is transcribed, but removed from within the transcript by splicing together the sequences (exons) on either side of it .

exon : region of genome that codes for portion of spliced mRNA; may contain 5'UTR, all CDSs, and 3' UTR.

CDS : only coding sequence; sequence of nucleotides of the exon that is protein coding (location includes stop codon).

terminator : sequence of DNA located either at the end of the transcript that causes RNA polymerase to terminate transcription.

5'UTR : region at the 5' end of a mature transcript (preceding the initiation codon) that is not translated into a protein.

5'clip : 5'-most region of a precursor transcript that is clipped off during processing.

3'UTR : region at the 3' end of a mature transcript (following the stop codon) that is not translated into a protein.

3'clip : 3'-most region of a precursor transcript that is clipped off during processing.

SIGNAL FEATURES

rep_origin : origin of replication; starting site for duplication of nucleic acid to give two identical copies.

enhancer : a cis-acting sequence that increases the utilization of (some) eukaryotic promoters, and can function in either orientation and in any location (upstream or downstream) relative to the promoter.

promoter : region on a DNA molecule involved in RNA polymerase binding to initiate transcription.

TATA_signal or **TATA_box :** Goldberg-Hogness box; a conserved AT-rich septamer found about 25 bp before the start point of each eukaryotic RNA polymerase II transcript unit which may be involved in positioning the enzyme for correct initiation. Consensus 'TATA(A or T)A(A or T)'.

polyA_site : site on an RNA transcript to which adenine residues by post-transcriptional polyadenylation will be added.

polyA_signal : recognition region necessary for endonuclease cleavage of an RNA transcript that is followed by polyadenylation. Consensus 'AATAAA'.

RBS : ribosome binding site.

-10_signal : Pribnow box; a conserved region about 10 bp upstream of he start point of bacterial transcription units which may be involved in binding RNA polymerase. Consensus 'TAtAaT'.

-35_signal : a conserved hexamer about 35 bp upstream of the start point of bacterial transcription units. Consensus 'TTGACa' or 'TGTTGACA'.

Repeat Features

repeat_region : region of genome containing repeating units.

repeat_unit : single repeat element.

satellite : many tandem repeats (identical or related) of a short basic repeating unit; many have a base composition or other property different from the genome average that allows them to be separated from the bulk (main band) genomic DNA.

LTR : long terminal repeat, a sequence directly repeated at both ends of a defined sequence.

STS : Sequence Tagged Site. Short, single-copy DNA sequence hat characterizes a mapping landmark on the genome and can be detected by PCR. A region of the genome can be mapped by determining the order of a series of STSs.

Here are some extra non-standard features —not listed in DDBJ/EMBL/GenBank tables— that may be useful working on genomic analysis:

Extra Gene Features

locus : "gene" in Mendel's sense.

orf : Open reading frame, sequence region between an Start and Stop signals.

transcript : A transcribed sequence region.

TSS : Transcription start site.

Start or **start_codon :** 'ATG' coding for initial methionine on nucleotide to protein sequence translation.

3'splice_site or **splice_donor** or **splice3' :** Intron start signal. Consensus 'GT'.

5'splice_site or **splice_acceptor** or **splice5' :** Intron end signal. Consensus 'AG'.

Extra Signal Features

CpG : CpG islands.

Regulatory_Reg : Regulatory regions.

SNP : Single Nucleotide Polymorphism.

TFBP : Regulatory protein region.

Appendix B : Reference Guide

This Appendix summarizes all the Command-line Options and those Variables which are available to improve your plots from Custom Files. When possible, all values for each option or variable are listed —including a short comment on it—; if not, you can find the reference for proper table on last Appendix where you can obtain those values.

- COMMAND-LINE **GFF2PS** GENERAL FORMAT.

Brackets are enclosing optional command-line elements, also output redirections are not mandatory —see figure 3.2 on section 3.2 for a basic idea— on the following Unix command-lines:

```
[cshell]$ ( gff2ps [-option [parameter] ... -option [parameter]] [--] \
          gff_file1 [... gff_filen] >output.ps ) >&output.rpt

[bshell]$ gff2ps [-option [parameter] ... -option [parameter]] [--] \
          gff_file1 [... gff_filen] >output.ps 2>output.rpt
```

- CUSTOM-FILE VARIABLES GENERAL FORMAT.

General format for Custom File Variables is explained in section 4.1.

For Layout features:

<*variable_name*>=<*value*>

For GFF-element, Group and Source features:

<*plot_element*>**::**<*variable_name*>=<*value*>

See section 4.3 for extra information on Command-line options, and section 4.2 to know more about variables in Custom Files.

B.2 Layout Features

Page Layout

page_size	**a4**	Available values are shown in table C on 'Page Format' column.
page_bbox	**auto,0,0**	Do not edit unless you want to define a new page size (overrides 'page_size'.
page_orientation	**Landscape**	'Landscape' or 'Portrait'
margin_left	**1cm**	You can use '#cm', '#in', '#pt'. '#' means any real number.
margin_right	**1cm**	
margin_upper	**1cm**	
margin_lower	**1cm**	
foreground_color	**FGcolor**	'FGcolor','BGcolor', or any of the color names defined on table C.6.
background_color	**BGcolor**	'FGcolor','BGcolor', or any of the color names defined on table C.6.
page_number	**1**	Any positive integer value greater or equal than 1.
zoom	**$*..*$**	'$*..*$', '$*$..end', 'start..$*$', 'start..end'. '$*$' means read nucleotide position from input. 'Start' and 'end' are nucleotide positions.
blocks_x_page	**1**	Any positive integer value, although more than five blocks in an A4 page looks crowded.
nucleotides_x_line	**0**	Default value '0' forces to calculate the sequence length from data. Any integer value means show those nucleotide number per page.
block_style	**default**	'default' —no framed blocks— or 'boxed' —put a frame around blocks—.
default_block_spacing_width	**0.25cm**	You can use '#cm', '#in', '#pt'. '#' means any real number.
show_blocks_top-bottom	**1**	Boolean switch: '1' means show blocks Top-to-Bottom, '0' means Left-to-Right.

Title Area

header_style	**default**	'none' —does not plot header area—, 'default' —no framed header—, 'boxed' —put a frame around header area—. Header elements are title, sub-title, page-number, date and time.
title	**default**	'none' —does not print title—, 'default' —prints first input filename—, 'free-text_string' —a title string given by user—.
subtitle	**default**	'none' —does not print sub-title—, 'default' —as 'none'—, 'free-text_string' —a sub-title string given by user—.
show_page_numbers	**on**	Boolean switch for displaying or not page numbering.
show_date	**on**	Boolean switch for displaying or not system date.
show_time	**on**	Boolean switch for displaying or not system time.

Tick Marks

major_tickmarks_num	**10**	Any positive integer number.
major_tickmarks_nucleotides	**-1**	Nucleotides between two neighbor major tickmarks, '-1' force program to use 'major_tickmarks_num' on 'nucleotides_x_line'
minor_tickmarks_num	**10**	Any positive integer number.
minor_tickmarks_nucleotides	**-1**	Nucleotides between two neighbor minor tickmarks, '-1' force program to use 'minor_tickmarks_num' on 'nucleotides_x_line'
show_grid	**on**	Boolean switch for plotting vertical dotted grid: '1' enables, '0' disables.
show_inner_scale	**both**	'none' disables any inner nucleotide ruler, 'both' shows top and bottom rulers, 'top' disables bottom and 'bottom' disables top ruler.
show_outer_scale	**both**	'none' disables any outer nucleotide ruler, 'both' shows top and bottom rulers, 'top' disables bottom and 'bottom' disables top ruler.
default_scale_width	**0.25cm**	You can use '#cm', '#in', '#pt'. '#' means any real number.
default_scale_spacing_width	**1**	Times that is multiplied **default_scale_width** to get the space between nucleotide rulers and source tracks.
nucleotide_scale	**default**	To set the units to show in the nucleotide rulers, 'default' or 'b' for nucleotide number, 'kb' for kilobases and 'mb' for megabases.

Block Layout

strand_show_forward	**on**	Boolean switch for displaying forward-strand area: 'on' enables, 'off' disables.
strand_show_reverse	**on**	Boolean switch for displaying reverse-strand area: 'on' enables, 'off' disables.
strand_show_independent	**on**	Boolean switch for displaying no-strand area: 'on' enables, 'off' disables.
show_left_source_label	**true**	Boolean switch, shows source labels at left side: 'true' enables, 'false' disables.
left_source_label_width	**2cm**	Left source labels width: You can use '#cm', '#in', '#pt'. '#' means any real number.
show_right_source_label	**false**	Boolean switch, shows source labels at right side: 'true' enables, 'false' disables.
right_source_label_width	**2cm**	Right source labels width: You can use '#cm', '#in', '#pt'. '#' means any real number.
sort_tracks_by_sequence	**on**	Boolean switch: 'on' to sort tracks by sequence names (comparing sources), 'off' to sort by source names (comparing sequences).
default_track_width	**1cm**	Default source tracks 'vertical' width: You can use '#cm', '#in', '#pt'. '#' means any real number.
default_track_spacing_width	**0.25cm**	Default source tracks spacer 'vertical' width: You can use '#cm', '#in', '#pt'. '#' means any real number.

General Definitions for Groups and GFF-Elements

frame_unknown_color	**orange**	Any of the color names defined on table C.6.
frame0_color	**blue**	
frame1_color	**red**	
frame2_color	**green**	
group_label_scale	**1**	Increasing or reducing group labels size by 'factor'.
position_label_scale	**1**	Increasing or reducing coords labels size by 'factor'.
show_positions	**false**	'true' or 'false'. Shows nucleotide start-end positions —in forward coordinates— for all elements on plot.
min_group_separation	**10**	Number of nucleotides from 0 to n that sets minimum nucleotide distance among two groups to avoid overlapping.

B.3 GFF-element Features

GFF-element Features

feature_color	**default**	'default', 1_color⇒'color', 2_color⇒'color..color', 3_color⇒'color..color..color'. Where each color name comes from table C.6.
feature_stroke_color	**default**	'default' for foreground color or any color name from table C.6.
shape	**box**	See table C.1 for available values.
fill_shape_mode	**default**	See table C.4 for available values.
fill_vector_mode	**default**	See table C.5 for available values.
vert_align	**default**	Available values are only 'default'/'center', or 'mirror'(from table C.2).
layer	**0**	first sorting for feature lines is made on FT_PROP["layer"]
label	**++default++**	(Not defined yet) '++none++' —does not display element label—, '++default++' —shows GFF-feature value—, ' "user-def" ' —free-text user defined label—.
show_feature	**on**	Boolean switch to avoid visualizing any feature: 'on' shows feature, 'off' does not show feature.
show_feature_positions	**off**	Boolean switch to show elements start-end nucleotide positions.

B.4 Group Features

feature_color	**default**	'default', 1_color⇒'color', 2_color⇒'color..color', 3_color⇒'color..color..color'. Where each color name comes from table C.6.
feature_stroke_color	**default**	'default' for foreground color or any color name from table C.6.
group_color	**default**	'default', 1_color⇒'color', 2_color⇒'color..color', 3_color⇒'color..color..color'. Where each color name comes from table C.6.
group_shape	**none**	See table C.1 for available values.
fill_shape_mode	**default**	See table C.4 for available values.
fill_vector_mode	**default**	See table C.5 for available values.
group_line	**default**	See table C.3 for available values.
group_line_color	**default**	Any of the color names defined on table C.6.
label	**++default++**	'++none++' —does not display group label—, '++default++' —shows group value—, ' "user-def" ' —free-text user defined label—.
vert_align	**default**	Available values are 'default'/'center', or 'mirror'(from table C.2).
show_group	**on**	Boolean switch to avoid visualizing any group: 'on' shows group, 'off' does not show group.
show_group_positions	**off**	Boolean switch to show, for all elements on that group, the start-end nucleotide positions.

B.5 Source Features

feature_color	**default**	'default', 1_color⇒'color', 2_color⇒'color..color', 3_color⇒'color..color..color'. Where each color name comes from table C.6.
feature_stroke_color	**default**	'default' for foreground color or any color name from table C.6.
group_color	**default**	'default', 1_color⇒'color', 2_color⇒'color..color', 3_color⇒'color..color..color'. Where each color name comes from table C.6.
left_label **right_label**	**++default++** **++default++**	'++none++' —does not display source label—, '++default++' —shows source or sequence name—, '++source++' —shows source name—, '++sequence++' —shows sequence name—, '++both++' —source & sequence names—, '++info++' —as previous plus grouping and strand—, ' "user-def" ' —free-text user defined label—.
show_left_label **show_right_label**	**on** **off**	Boolean switch to show or not the left/right track labels area.
source_label_scale	**1**	Increasing or reducing source labels size by 'factor'.
source_style	**default**	'default' —no framed source-track— or 'boxed' —put a frame around source-track—.
source_line	**default**	See upper panel from table C.3 for available values.
source_line_color	**red**	Any of the color names defined on table C.6.
vert_align	**default**	See table C.2 for available values.
range	**default**	Defining range of lower-upper source-scores: 'none' —all scores are set to '1'—, 'default' —score range got from input files—, '#..#' —forcing range to defined values—.
track_scale	**1**	Any positive real number, this variable allows you to resize one or more source-tracks.
track_spacing_scale	**1**	Any positive real number, this variable allows you to resize one or more source-track spacers.
keep_feature_label_space	**on**	'on' reserves half of the source track width for labels, 'off' forces the features to be drawn fitting the full source track width.
unfold_grouped_ungrouped	**off**	Boolean switch to split grouped elements from ungrouped: 'on' grouped and ungrouped elements shown in two tracks, 'off' all grouped and ungrouped elements fitting only one track.
unfold_grouped_line	**on**	Boolean switch to split overlapping grouped elements in several tracks: 'on' split groups, 'off' all grouped elements fitting only one track.
unfold_ungrouped_line	**on**	Boolean switch to split tracks for overlapping ungrouped elements: 'on' split ungrouped elem., 'off' all ungrouped elem. to one track.
show_source_positions	**off**	Boolean switch to show, for all elements on that source, the start-end nucleotide positions.

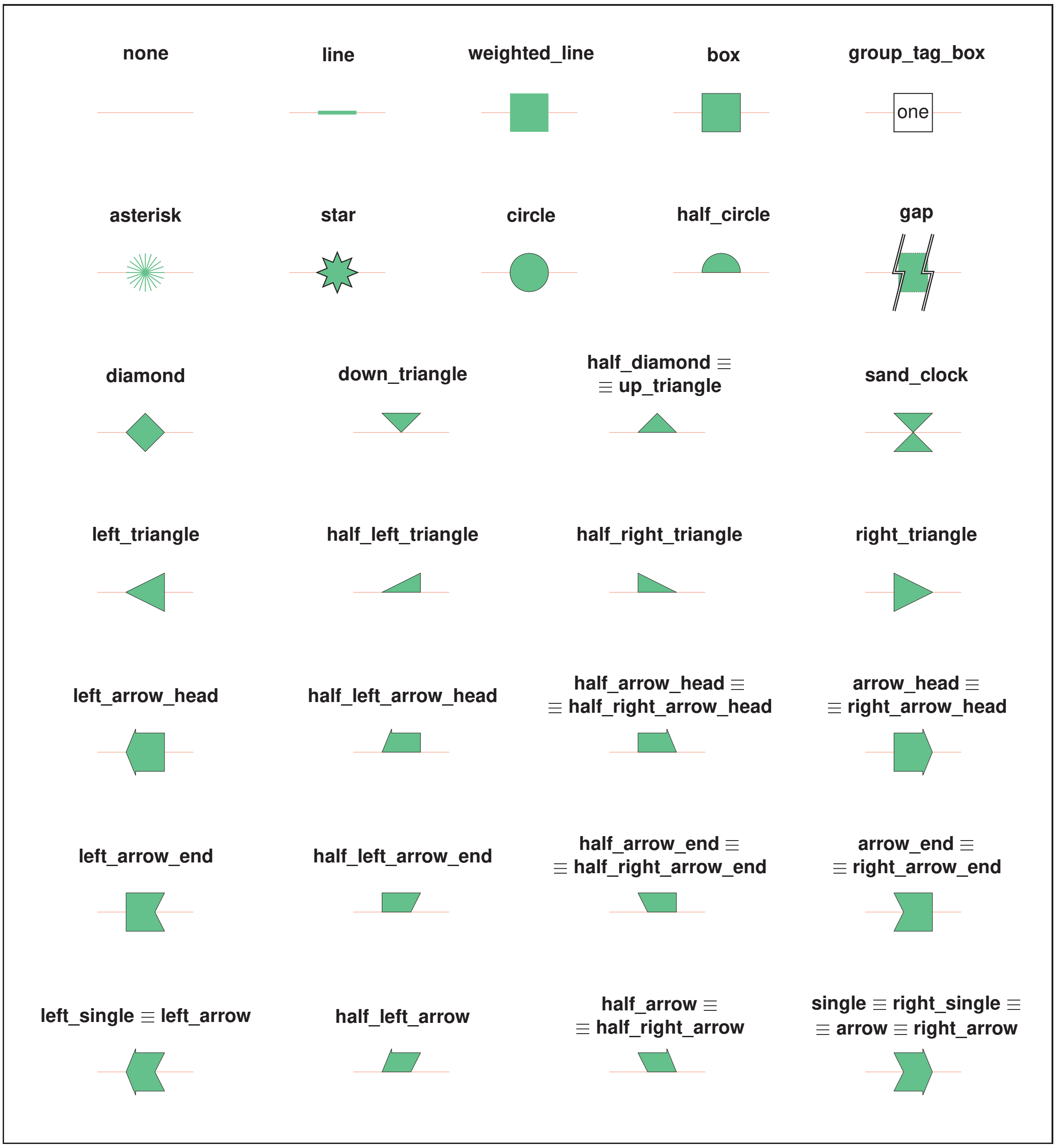
B.1 Shell command-line options for gff2ps.

-h	Shows command-line help.
-H <*option*>	Shows only help for the especified option.
-V	Verbose mode, a full report is sent to standard error (default only sends Warnings).
-v	Quiet mode: Disable warnings, no messages sent to standard error.
-d	Write (or rewrite if exists) default customfile (.gff2psrc).
-D <*default_cf*>	Create a new default customfile with the given filename.
-C <*c_filename*>	Load given custom file and append to default custom file.
-s <*page_size*>	Useful to modify the page size (default is a4).
-p	Switches page orientation to Portrait (default is Landscape).
-G <*color*>	Sets color for FOREGROUND (default is black).
-g <*color*>	Sets color for BACKGROUND (default is white).
-P <*#*>	Sets how many pages are needed to split your output (default is one).
-S <*#*>	Zoom first nucleotide (default is sequence origin).
-E <*#*>	Zoom last nucleotide (default is sequence length).
-B <*#*>	Sets blocks per page (default is one).
-N <*#*>	Sets nucleotides per line (default is the largest sequence position from input gff-files).
-b	Blocks from left to right and from top to bottom (default is top to bottom first).
-L	Switch off Header (Title area).
-T <*quoted_string*>	Defining title (default is input gff filename).
-t <*quoted_string*>	Defining subtitle (default is none).
-l	Does not show page numbering.
-O	Does not show date.
-o	Does not show time.
-M <*#*>	Number of major tickmarks per line (default 10).
-K <*#*>	Major tickmarks scale in nucleotides, default is nucleotide length for lines divided by major tickmarks number (see option **-M**).
-m <*#*>	Number of minor tickmarks between major tickmarks (default 10).
-k <*#*>	Minor tickmarks scale in nucleotides default is major tickmarks size divided by minor tickmarks number (see option **-m**).
-w *or* **-f**	Switch off displaying forward-strand (Watson) elements.
-c *or* **-r**	Switch off displaying reverse-strand (Crick) elements.
-i	Switch off displaying strand-independent elements.
-0 <*color*>	Sets color for frame ‘0’ (default is blue).
-1 <*color*>	Sets color for frame ‘1’ (default is red).
-2 <*color*>	Sets color for frame ‘2’ (default is green).
-3 <*color*>	Sets color for frame ‘.’ (default is orange).
-n	Switch off labels for element positions.
-a	Disables copyright line printing.

<*#*>	An integer value.
<*quoted_string*>	A free text string between double quotes (‘"’).
<*page_size*>	See available values on Appendix C.7 table.
<*color*>	A color name chosen from table on Appendix C.6.
<*default_cf*>	New name for Default Custom File, see Section 4.2.
<*c_filename*>	Name for user-defined Custom File, see also Section 4.2.

Appendix C : Options Summary

Table C.1: Available shape names.



Each shape is defined by <*start*> and <*end*> nucleotide positions, 'default' track vertical alignment was used to obtain this figure (see figure C.1). Its 'vertical' width is defined by <*score*>, except 'none', 'default' and 'line', which have no width. Red line represents in this table source baseline, and all shapes are centered on it. '≡' means that names are equivalent (synonymous). 'default' is set in different ways for groups and GFF-features, while in groups means same as 'none', for GFF-features is defined as 'box'. For all the shapes 'green' fill color was defined in the variable '**feature_color**', while 'black' color was defined in the variable '**feature_stroke_color**'.

Table C.2: Available baseline alignment (Forward and No-Strand elements).

VARIABLE VALUE	SOURCES	GFF-ELEMENTS & GROUPS
reverse ≡ top	Source-Track	
default ≡ center	Source-Track	
mirror	Source-Track	
baseline ≡ bottom	Source-Track	

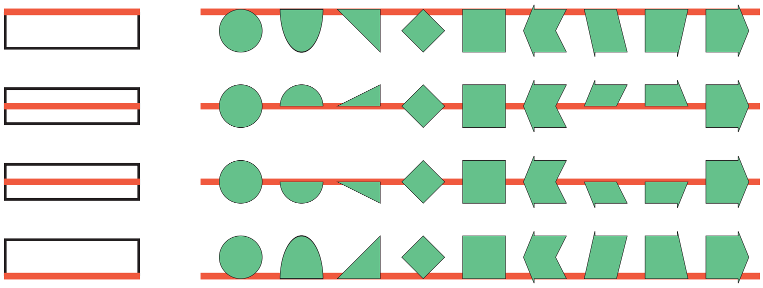
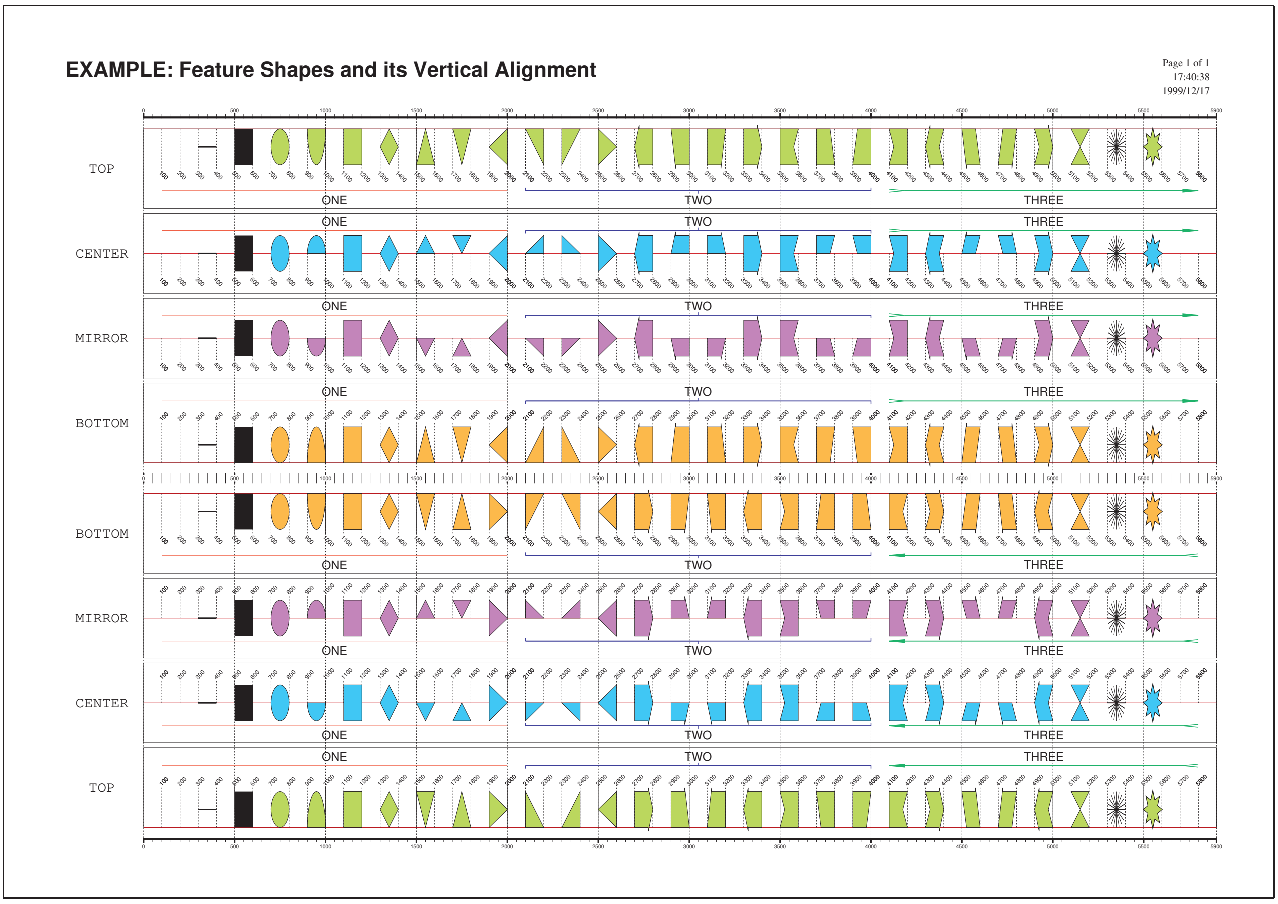


Figure C.1: Showing track vertical alignment for sources.



gff2ps mirrors reverse strand source order from forward sources. The following plot shows how this mirroring affects source vertical alignment. You can also see nucleotide start and end positions for each feature. You can get more examples from:

`http://www1.imim.es/~jabril/GFFTOOLS/GFF2PS-Snapshots.html`

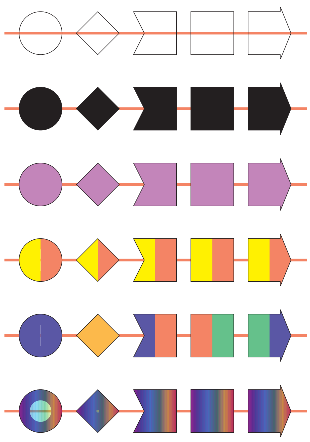
Table C.3: Available line-types.

VARIABLE VALUE	OUTPUT
none	
default ≡ line	
dotted ≡ dotted_line	
long_dotted	
short_dashed	
long_dashed	
Group features extra line-types	
bracket	
arrow ≡ right_arrow	
left_arrow	

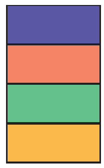
Table C.4: Available fill-shape modes

SHAPE-MODE VARIABLE VALUES fill_shape_mode	feature_color
none	—
default	—
1_color	violet
2_color	yellow..red
frame-remainder	—
rainbow	—

OUTPUT



DEFAULT COLORS FOR **FRAME-REMAINDER** MODE.



frame0	remainder0
frame1	remainder2
frame2	remainder1
no-frame	no-remainder

Those values can be modified redefining following variables: 'frame0_color', 'frame1_color', 'frame2_color' and 'frame_unknown_color'.

Table C.5: Available fill-vector modes

VECTOR-MODE VARIABLE VALUES fill_vector_mode	feature_color
none	—
default	—
1_color	violet
2_color	yellow..red
3_color	red..green..blue
rainbow	—

OUTPUT

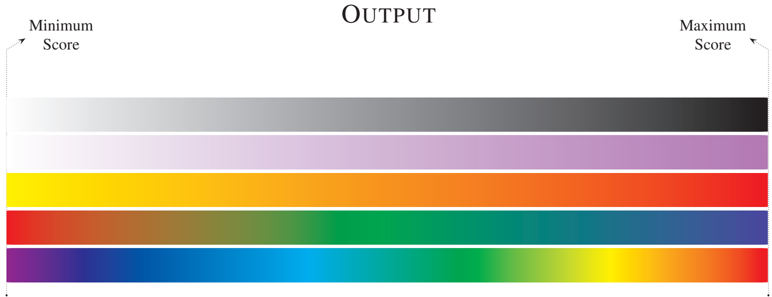


Table C.6: **`gff2ps`** CMYK color definition table and Color Names.

black	0.00	0.00	0.00	1.00
verydarkgrey	0.00	0.00	0.00	0.80
darkgrey	0.00	0.00	0.00	0.60
grey	0.00	0.00	0.00	0.40
lightgrey	0.00	0.00	0.00	0.20
verylightgrey	0.00	0.00	0.00	0.10
white	0.00	0.00	0.00	0.00
verydarkmagenta	0.00	1.00	0.00	0.30
darkmagenta	0.00	0.80	0.00	0.05
magenta	0.00	0.60	0.00	0.00
lightmagenta	0.00	0.40	0.00	0.00
verylightmagenta	0.00	0.20	0.00	0.00
verydarkviolet	0.45	0.85	0.00	0.00
darkviolet	0.30	0.65	0.00	0.00
violet	0.22	0.55	0.00	0.00
lightviolet	0.15	0.40	0.00	0.00
verylightviolet	0.10	0.20	0.00	0.00
verydarkblue	1.00	1.00	0.00	0.20
darkblue	0.90	0.90	0.00	0.00
blue	0.75	0.75	0.00	0.00
lightblue	0.50	0.50	0.00	0.00
verylightblue	0.30	0.30	0.00	0.00
verydarkskyblue	0.90	0.50	0.00	0.15
darkskyblue	0.75	0.45	0.00	0.00
skyblue	0.60	0.38	0.00	0.00
lightskyblue	0.45	0.25	0.00	0.00
verylightskyblue	0.30	0.15	0.00	0.00
verydarkcyan	1.00	0.00	0.00	0.10
darkcyan	0.80	0.00	0.00	0.00
cyan	0.60	0.00	0.00	0.00
lightcyan	0.40	0.00	0.00	0.00
verylightcyan	0.20	0.00	0.00	0.00

verydarkseagreen	0.75	0.00	0.45	0.00
darkseagreen	0.62	0.00	0.38	0.00
seagreen	0.50	0.00	0.30	0.00
lightseagreen	0.38	0.00	0.22	0.00
verylightseagreen	0.25	0.00	0.15	0.00
verydarkgreen	1.00	0.00	1.00	0.25
darkgreen	0.80	0.00	0.80	0.00
green	0.60	0.00	0.60	0.00
lightgreen	0.40	0.00	0.40	0.00
verylightgreen	0.20	0.00	0.20	0.00
verydarklimegreen	0.50	0.00	1.00	0.10
darklimegreen	0.40	0.00	0.95	0.00
limegreen	0.30	0.00	0.80	0.00
lightlimegreen	0.20	0.00	0.65	0.00
verylightlimegreen	0.10	0.00	0.50	0.00
verydarkyellow	0.00	0.00	1.00	0.25
darkyellow	0.00	0.00	1.00	0.10
yellow	0.00	0.00	1.00	0.00
lightyellow	0.00	0.00	0.50	0.00
verylightyellow	0.00	0.00	0.25	0.00
verydarkorange	0.00	0.50	0.80	0.10
darkorange	0.00	0.40	0.80	0.00
orange	0.00	0.30	0.80	0.00
lightorange	0.00	0.20	0.75	0.00
verylightorange	0.00	0.15	0.70	0.00
verydarkred	0.00	1.00	1.00	0.15
darkred	0.00	0.80	0.80	0.00
red	0.00	0.60	0.60	0.00
lightred	0.00	0.40	0.40	0.00
verylightred	0.00	0.20	0.20	0.00
verydarkbrown	0.35	0.85	1.00	0.40
darkbrown	0.30	0.70	1.00	0.35
brown	0.25	0.75	1.00	0.25
lightbrown	0.20	0.60	0.70	0.15
verylightbrown	0.15	0.45	0.55	0.00

Table C.7: Page Sizes defined in **`gff2ps`**.

PAGE FORMAT	PAGE SIZE (in points)		(in cms)		(in inches)	
a0	2384	3370	84.1	118.9	33.1	46.8
a1	1684	2384	59.4	84.1	23.4	33.1
a2	1190	1684	42.0	59.4	16.5	23.4
a3	842	1190	29.7	42.0	11.7	16.5
a4	595	842	21.0	29.7	8.3	11.7
a5	420	595	14.8	21.0	5.8	8.3
a6	297	420	10.5	14.8	4.1	5.8
a7	210	297	7.4	10.5	2.9	4.1
a8	148	210	5.2	7.4	2.1	2.9
a9	105	148	3.7	5.2	1.5	2.1
a10	73	105	2.6	3.7	1.0	1.5
b0	2920	4127	103.0	145.6	40.6	57.3
b1	2064	2920	72.8	103.0	28.7	40.6
b2	1460	2064	51.5	72.8	20.3	28.7
b3	1032	1460	36.4	51.5	14.3	20.3
b4	729	1032	25.7	36.4	10.1	14.3
b5	516	729	18.2	25.7	7.2	10.1
b6	363	516	12.8	18.2	5.0	7.2
b7	258	363	9.1	12.8	3.6	5.0
b8	181	258	6.4	9.1	2.5	3.6
b9	127	181	4.5	6.4	1.8	2.5
b10	91	127	3.2	4.5	1.3	1.8
executive	540	720	19.0	25.4	7.5	10.0
folio	612	936	21.6	33.0	8.5	13.0
legal	612	1008	21.6	35.6	8.5	14.0
letter	612	792	21.6	27.9	8.5	11.0
quarto	610	780	21.5	27.5	8.5	10.8
statement	396	612	14.0	21.6	5.5	8.5
10x14	720	1008	25.4	35.6	10.0	14.0
ledger	1224	792	43.2	27.9	17.0	11.0
tabloid	792	1224	27.9	43.2	11.0	17.0
userdefined	595	2384	21.0	84.1	8.3	33.1