Implementation of a standalone Galaxy instance for the creation of re-usable bioinformatics processing pipelines

Daniel J. Shea

August 16th, 2014

1 Introduction

Using yeast (Schizosaccharomyces Pombe and Saccharomyces Cervisiae) as our model organism, our research is conducted into understanding the biological processes underlying the formation, function, and epigenetic inheritance of silent chromatin domains. Three major areas of research are related to RNAi-mediated assembly of heterochromatin, SIR-mediated assembly of silent chromatin, and the regulation of recombination within the ribosomal DNA repeats. In order to improve the efficiency of data analysis via re-usable bioinformatics pipelines and to encourage collaboration and software re-use within the lab, we have chosen to implement a local instance of the Galaxy Project's open source web-based platform.

1.1 Heterochromatin's role in the silencing of genes

The physical organization of DNA in eukaryotes plays a role in the levels of genetic expression by restricting the physical access of transcriptional machinery to various regions of an organism's genome. The organization of DNA is structured around histones, octamers composed of highly conserved proteins, H2A, H2B, H3, and H4. These proteins form heterodimers H2A/H2B and H3/H4, respectively, that then further form tetramers. In turn, these tetramers then combine to form an octamer that is referred to as a histone. DNA winds around these histones forming a nucleosome, with small segments of DNA, referred to as linker DNA, between them. Structured around this basic organization of DNA are regions of tightly or loosely condensed chromatin, referred to as heterochromatin and euchromatin respectively. Heterochromatin plays a functional role in gene silencing through the physical restriction of access to DNA by transcriptional machinery involved in transcribing DNA into messenger RNA (mRNA).

1.2 Benefits derived from the implementation of Galaxy within the lab

Galaxy is an open, web-based platform for the analysis of data intensive biological research. It provides an interactive and readily accessible web interface to many of the commonly used bioinformatics tools available today. Computational pipelines may be constructed from within the web interface and then shared between users. Work flows are accessible via the work flow editor, which allows for the visualization and editing of processing pipelines. These work flows may then be abstracted for application to multiple data sets, and the work flows themselves can be shared between Galaxy users through the Galaxy web interface. This allows researchers within our environment to compose work flows that may then be re-used or re-purposed by other collaborators in the lab with minimal effort.

2 Computational server architectural overview

Our computational server, `chromosome.med.harvard.edu`, serves as the platform upon which we run our Galaxy instance. It should be noted however, that a local galaxy instance can be run on hardware with relatively modest specifications. Depending upon the data sets and types of analyses performed, you may be constrained by the available amount RAM on your server when dealing with particularly large data sets or application of computationally intensive analysis such as attempting to perform de novo assembly. The table below details the hardware configuration for chromosome.

2.1 Hardware configuration

	chromosome.med.harvard.edu
CPU:	(2) Intel(R) Xeon(R) CPU X5660 @ 2.80GHz, 6 Physical cores (12 Hyper-threaded)
Memory:	(18) 4GB DDR3-800 DIMMs for a total of 72GB
Disk:	(8) 1.817 TB SATA Drives configured as a RAID-6 array, providing 10.908TB of usable space
OS:	Ubuntu 12.04.4 LTS

3 Galaxy architectural overview

Galaxy provides default web server and local database storage for standalone instances. Depending on the use case, it is advised to reconfigure the instance to make use of more robust data storage and/or web server solutions. For our particular instance, we are currently utilizing a postgresql database as our back-end storage and have chosen to utilize the proxy capability of nginx to serve website content. By default, sqlite internal database files are utilized for the storage of data and computational work flows. This proved problematic as our anticipated data sets were determined to quickly tax a small flat file database system such as sqlite. Therefore, we opted to create a postgresql database wherein to store all of our data. Details on how to accomplish this are detailed below in the database subsection. Details on configuration of nginx as a proxy are outlined in the nginx configuration subsection.

User authentication and role based authentication are handled by the Galaxy server itself. Larger installations with access to a centralized user identification management system such as LDAP or kerberos may be able to take advantage of identity caching through configuration of login caching and forwarding via apache or nginx. Attempts to tie our local instance to existing LDAP authentication mechanisms proved to break the existing administrative functionality of the server. These issues were raised with the development team of the Galaxy Project and it is currently being assessed as a possible feature request by Galaxy development team.

Below, we outline the necessary steps undertaken to install and configure our Galaxy standalone instance using an nginx proxy-based web server and a postgresql back-end database for the storage of our data.

3.1 Galaxy, postgresql and nginx installation

To install Galaxy, we can retrieve the latest stable version from the mercurial repository. In order to do this, we must have the mercurial package installed. *(**Note:** Since we are using Ubuntu server as our linux distribution, we will demonstrate how to do this using the Ubuntu package management tools. Please refer to your operating system's package management documentation if you are using a different distribution of linux.)*

```
sudo apt-get install mercurial
```

Once mercurial is installed we may then install the latest stable Galaxy from the mercurial repository.

```
hg clone https://bitbucket.org/galaxy/galaxy-dist/
cd galaxy-dist
hg update stable
```

This is enough to configure a simple standalone Galaxy instance utilizing the internal sqlite database and python webserver. You can test the installation by attempting to start Galaxy by running the following command from within the `galaxy-dist` directory.

```
sh run.sh
```

The initial run of Galaxy will pull necessary python .egg files and start a local instance of the Galaxy server. We can now shut down the test instance as we will next install postgresql and configure Galaxy to rely on the external database as our back-end data store for all of our research data. Shutdown of Galaxy may be performed by issuing a `Ctrl-c` in the terminal where `sh run.sh` was instantiated.

To configure postgresql we will need to install the postgresql package.

```
sudo apt-get install postgresql
```

Further details regarding the configuration of postgresql may be obtained in your operating system's system administration documentation, or by accessing postgresql documentation available online. Once the database software is installed, we must create a database owned by the user id that will run the Galaxy server. In our case the user id is `galaxy`. *(**Note:** The name of the database user created in postgresql should match whatever username is chosen.)* Since the Galaxy platform controls its own schema within the database, we need to ensure that our galaxy user has the appropriate permissions and ownership of the the database for table creation. This is controlled through the postgresql configuration. Instructions on how to assign roles and permissions to a postgresql user may be referenced in the postgresql documentation.

The nginx webserver may be installed by issuing the following command.

```
sudo apt-get install nginx
```

For more administrative configuration details concerning nginx, please look at the nginx documentation. We will breifly outline the steps we have taken to set up nginx as a proxy for Galaxy, however this is only one possible configuration. Further details regarding use of nginx or apache may be referenced in the Galaxy documentation.

3.2 Integration of postgresql with Galaxy

Once a database has been created and we have created both the relevant galaxy user in the operating system and in the database authentication systems, we can begin to configure Galaxy for use with an external database. Galaxy stores its relevant configuration information in a file located at the top-level directory `galaxy-dist` within a file named `universe_wsgi.ini`.

First we should ensure that the Galaxy server will be accessible from external interfaces if we are planning on setting up a web accessible standalone instance. This can be done by making the following change to the host entry within `universe_wsgi.ini`:

```
# The address on which to listen.  By default, only listen to
# localhost (Galaxy will not be accessible over the network).
# Use '0.0.0.0' to listen on all available network interfaces.
# host = 127.0.0.1}
host = 0.0.0.0
```

The next line we will need to edit is the `database_connection` line. This will point Galaxy to our newly created postgresql database.

```
# Connect to a local postgresql database where database galaxy has been configured for
# access by the galaxy user. - djs 2014-05-06
# The SQLAlchemy PostgreSQL dialect has been renamed from 'postgres' to 'postgresql'.
# The new URL format is postgresql[+driver]://<user>:<pass>@<host>/<dbname>
database_connection = postgresql://galaxy:PASSWORD_GOES_HERE@localhost/galaxy
```

We will also enable the optional server side cursors which will improve the performance on large queries. This option is only available if we are using postgresql and was a consideration when determining which database to use as our storage model.

```
# If large database query results are causing memory or response time issues in
# the Galaxy process, leave the result on the server instead.  This option is
# only available for PostgreSQL and is highly recommended.
#database_engine_option_server_side_cursors = False
database_engine_option_server_side_cursors = True
```

The next three entries into our configuration will direct Galaxy where to manage tools for the tool shed repositories and where dependencies are to be stored on the server.

```
# Tool config files, defines what tools are available in Galaxy.
# Tools can be locally developed or installed from Galaxy tool sheds.
#tool_config_file = tool_conf.xml,shed_tool_conf.xml
tool_config_file = tool_conf.xml,shed_tool_conf.xml
```

```
# Default path to the directory containing the tools defined in tool_conf.xml.
# Other tool config files must include the tool_path as an attribute in the <toolbox>
    tag.
#tool_path = tools
tool_path = tools
```

```
# Path to the directory in which managed tool dependencies are placed.  To use
# the dependency system, see the documentation at:
# http://wiki.g2.bx.psu.edu/Admin/Config/Tool%20Dependencies
tool_dependency_dir = tool_dependencies
```

Next, we will need to configure Galaxy to take advantage of Data Libraries, a method for sharing data within the lab. Directories in the configuration file listed here are relative paths, meaning, they are assumed to exist underneath our local `galaxy-dist` directory. It should be noted that if you define a directory within the configuration file, you must still create the directory yourself and ensure it has appropriate permissions for access by Galaxy. in our case the owner of the Galaxy processes is the `galaxy` user, so we created the relevant directories and ensured that the ownership was set appropriately.

Below is the entry in the configuration file telling Galaxy where files will be placed in order for the Galaxy administrator to upload them into Galaxy.

```
# -- Data Libraries

# These library upload options are described in much more detail in the wiki:
# http://wiki.g2.bx.psu.edu/Admin/Data%20Libraries/Uploading%20Library%20Files

# Add an option to the library upload form which allows administrators to
# upload a directory of files.
library_import_dir = MoazedLabData/Processed
```

And here is the relevant system directory, showing the appropriate permissions for that directory.

```
root@chromosome:/usr/local/galaxy-dist# ls -ld MoazedLabData/
drwxr-xr-x 4 galaxy galaxy 89 Jun 12 14:22 MoazedLabData/
```

We will now enable the functionality that allows us to paste system specific paths via the Data Library management tool. Since our galaxy user is restricted to the Galaxy installation itself, this does not pose a security risk. However, it is strongly advised that you carefully consider the implications of enabling this feature if you choose to run galaxy with a normal account. It is advised that you create a galaxy account for running galaxy and restrict the galaxy user to only allow access to relevant system files. Please refer to your operating system's system administration documentation if you are unsure how to do this, as it is outside the scope of this article.

```
# Add an option to the admin library upload tool allowing admins to paste
# filesystem paths to files and directories in a box, and these paths will be
# added to a library.  Set to True to enable.  Please note the security
# implication that this will give Galaxy Admins access to anything your Galaxy
# user has access to.
allow_library_path_paste = True
```

Before we open up our Galaxy instance to users, it is very important that we change the default `id_secret` in the configuration file. Failing to do so poses a security risk. This value should be some random string, in this example I have generated a randomized string of data for use as the secret using the apg utility. If you do not have the apg utility installed on your system, in Ubuntu, it can be added by issuing the following command:

```
sudo apt-get install apg
```

Now we will want to use apg to generate our `id_secret`. This can be done by issuing the following command:

```
apg -a 1 -m 32 -x 32
```

`-a` tells `apg` to use random characters instead of attempting to provide a pronounceable password. `-m` and `-x` are flags indicating the minimum and maximum lengths of the generated passwords. Here we have chosen 32 for both values so that all passwords generated will be 32 characters in length.

Below is a sample output from the above referenced command.

```
]d_@{MF+cgItXmc+s[{J64eAKQKB{/6U
~D!N$M(\oF%*q1I!WSrfEydZ6MaK.-.`
g~V?"A\=,T.,9hi:]4e]<$8~y`B\+uJJ
lA=ivi3>GF09OB9qNbeGaDhp#eaO(q%j
t=Y!l_Ip^NsT!46U+>tJ##^ug($H{en<
9K%hA=P]F^x[5M@n2tp$S(ir-akB-%yQ
```

Each line represents a randomly generated 32 character password. Select one and place that string in the `id_secret` entry within the `universe_wsgi.ini` configuration file as shown below. This key is used by Galaxy to generate internal values involved with cookies and URLs. It must be changed or your Galaxy installation will not be secure from malicious users who could generate fake cookies using the default entry to gain access to a user's Galaxy account and thereby their data!

```
# -- Users and Security

# Galaxy encodes various internal values when these values will be output in
# some format (for example, in a URL or cookie).  You should set a key to be
# used by the algorithm that encodes and decodes these values.  It can be any
# string.  If left unchanged, anyone could construct a cookie that would grant
# them access to others' sessions.
#id_secret = USING THE DEFAULT IS NOT SECURE!
id_secret = g~V?"A\=,T.,9hi:]4e]<$8~y`B\+uJJ
```

The next setting we may need to change is to normalize emails. Galaxy user ids are constructed from the user's email address given at registration time. Normalization will force the email address to all lowercase and avoid any potential case issues when a user enters their email address to login to the system.

```
# If your proxy and/or authentication source does not normalize e-mail
# addresses or user names being passed to Galaxy - set the following option
# to True to force these to lower case.
normalize_remote_user_email = True
```

It is important if you are planning on administrating your galaxy instance to add your own email address to this entry and to create a user with a matching email address. This will enable you to access the administrative functions later when bringing up the Galaxy instance and is important to be able to create the shared Data Libraries.

```
# Administrative users - set this to a comma-separated list of valid Galaxy
# users (email addresses).  These users will have access to the Admin section
# of the server, and will have access to create users, groups, roles,
# libraries, and more.  For more information, see:
# http://wiki.g2.bx.psu.edu/Admin/Interface
```

```
#admin_users = None
admin_users = your.email@university.edu
```

Here we disable anonymous login, as only registered users will be given access to the laboratory instance of Galaxy.

```
# Force everyone to log in (disable anonymous access).
require_login = True
```

Currently, we have left this as the default, allowing anyone to register an account with our instance. Since our instance is behind a firewall an only accessible by individuals with access to the Private HMS network, we have decided to let our users create their own accounts and specify the email they would like to use. Depending upon your network configuration and whether or not your Galaxy instance is accessible over the internet, you may wish to change this value to `False`.

```
# Allow unregistered users to create new accounts (otherwise, they will have to
# be created by an admin).
#allow_user_creation = True
```

We have enabled administrative functionality to delete accounts. By default it is not enabled. Due to our rather lenient account creation policy we felt it necessary should individuals accidentally create a second account for themselves.

```
# Allow administrators to delete accounts.
#allow_user_deletion = False
allow_user_deletion = True
```

3.3 Integration of nginx for use as a proxy with Galaxy

Nginx is a webserver software package that can be utilized as a proxy to serve static html and to provide improved server response in comparison to the built-in python webserver that Galaxy deploys natively. Here, we will detail the necessary configuration steps that were required to enable Galaxy to work with nginx as a proxy.

The configuration file for the nginx webserver are located in `/etc/nginx/nginx.conf`. This file will require the following changes in order for nginx to serve as a proxy for Galaxy. Within the `http` clause, the following settings will allow nginx to utilize gzip compression to reduce the amount of data that needs to be passed through the proxy. It is advisable to add this to the configuration if it is not already present as it significantly speeds up performance of the webserver when dealing with large data sets.

```
        ##
        # Gzip Settings
        ##

        gzip on;
        gzip_disable "msie6";

        gzip_vary on;
        gzip_proxied any;
        gzip_comp_level 6;
        gzip_buffers 16 8k;
        gzip_http_version 1.1;
        gzip_types text/plain text/css application/json application/x-javascript text/
    xml application/xml application/xml+rss text/javascript;
```

These filters must also be enabled in the Galaxy configuration file `universe_wsgi.ini` as well, as they are disabled by default. Here, the proxy filters are defined near the top of the configuration file and we have enabled them.

```
# ---- Filters --------------------------------------------------------------

# Filters sit between Galaxy and the HTTP server.

# These filters are disabled by default.  They can be enabled with
# 'filter-with' in the [app:main] section below.

# Define the gzip filter.
[filter:gzip]
use = egg:Paste#gzip

# Define the proxy-prefix filter.
[filter:proxy-prefix]
use = egg:PasteDeploy#prefix
prefix = /
```

Next, we comment out the entry for the default built-in webserver and enable filtering with a proxy.

```
# If not running behind a proxy server, you may want to enable gzip compression
# to decrease the size of data transferred over the network.  If using a proxy
# server, please enable gzip compression there instead.
#filter-with = gzip

# If running behind a proxy server and Galaxy is served from a subdirectory,
# enable the proxy-prefix filter and set the prefix in the
# [filter:proxy-prefix] section above.
filter-with = proxy-prefix
```

Back within the `/etc/nginx/nginx.conf` file, we will now define the Galaxy application and direct the proxy to the location of static files to be served. Since we are only running Galaxy on this server's web server, our configuration is rather minimal. We serve pages on port 8080 (the default Galaxy port) and point the aliases to the appropriate directories underneath the Galaxy base installation directory `/usr/local/galaxy-dist`.

```
        ##
        # Galaxy configuration
        ##
        upstream galaxy_app {
        server localhost:8080;
        }
        server {
                client_max_body_size 10G;
                # ... other server stuff ...
                location / {
                proxy_set_header X-Forwarded-Host $host;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-URL-SCHEME https;
                }
                location /static {
                alias /usr/local/galaxy-dist/static;
                }
                location /static/style {
                alias /usr/local/galaxy-dist/static/june_2007_style/blue;
                }
                location /static/scripts {
                alias /usr/local/galaxy-dist/static/scripts/packed;
                }
                location /favicon.ico {
                alias /usr/local/galaxy-dist/static/favicon.ico;
                }
                location /robots.txt {
                alias /usr/local/galaxy-dist/static/robots.txt;
                }
        }
```

Once you have finished configuration of your Galaxy instance you can again bring it up and test that the configuration settings are working as intended. You can verify that the Galaxy server is able to access your postgresql database and a new schema should appear within the database now that Galaxy is managing its internal storage via the postgresql database. There is an important log associated with startup that should be mentioned at this point, as it will assist in troubleshooting any potential configuration issues and will allow you to determine when Galaxy has successfully started.

4 Galaxy system service configuration

Since Galaxy is running as a system service, we will make several additional operating system level modifications to our server. The following subsections briefly outline such configuraton changes with respect to Ubuntu 12.04LTS. Postgresql and nginx should already be enabled as system services that start at boot time as part of their installation via the package manager, therefore we will only cover the necessary steps required to ensure that Galaxy is also enabled and brought up as a system service.

4.1 Galaxy log file rotation using logrotate

Located under the galaxy installation (in our case `/usr/local/galaxy-dist`) we will find a log file named `paster.log`. This file is where the Galaxy instance will log messages while it is running on the system.

On linux based systems, it might be beneficial to utilize `logrotate` to handle ensuring that `paster.log` is routinely rotated and compressed, to avoid continuously logging to a single file that will, over time, grow and consume system resources such as disk space. This can be done by implementing a logrotate configuration for galaxy by placing a file named `galaxy` in `/etc/logrotate.d` with the following configuration.

```
/usr/local/galaxy-dist/paster.log {
  weekly
  rotate 8
  copytruncate
  compress
  missingok
  notifempty
}
```

This tells the logrotate dæmon to rotate paster.log weekly, keeping the last 8 logs, copying the existing file and truncating the current log, and then compressing the copied logfile to reduce disk utilization storing the files. It is ok if the file is missing, and do not perform the above actions if the current log file is empty. For more information on how to configure the logrotate dæmon itself, please refer to your operating system administration documentation.

4.2 Automating start up and shut down of the Galaxy service

Since Galaxy will be treated as a system service on our server, we have opted to create a simple start up and shut down script on our Ubuntu server utilizing the Ubuntu upstart system. this ensures that when our system is brought down for maintenance, or re-started, that our Galaxy instance will be brought down cleanly during the shut down process and brought back up during the system's boot sequence. Below are the contents of our script for reference.

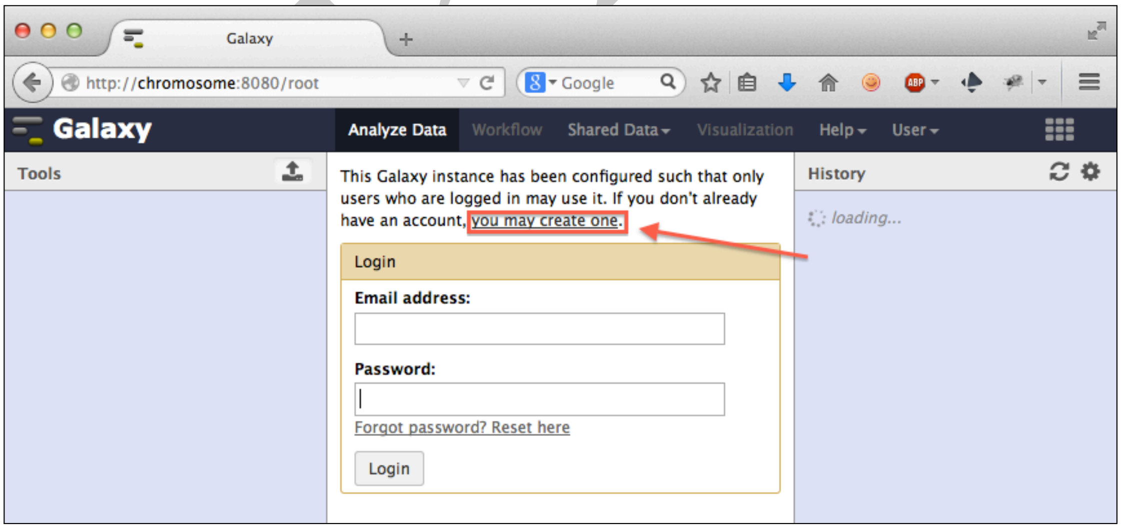
```
#!/bin/bash

### BEGIN INIT INFO
# Provides:          galaxy
# Required-Start:    $local_fs $remote_fs $syslog $named $network $time
# Required-Stop:     $local_fs $remote_fs $syslog $named $network
# Should-Start:
# Should-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start/Stop the Galaxy service
### END INIT INFO

#--- config
SERVICE_NAME=galaxy
RUN_AS=galaxy
RUN_IN=/usr/local/galaxy-dist
#--- main actions

start() {
        echo "Starting $SERVICE_NAME... "
        cmd="cd $RUN_IN && sh run.sh --daemon"
        case "$(id -un)" in
                $RUN_AS)
                        eval "$cmd"
                        ;;
                root)
                        su - $RUN_AS -c "$cmd"
                        ;;
                *)
                        echo "*** ERROR *** must be $RUN_AS or root in order to control
     this service" >&2
                        exit 1
        esac
        echo "...done."
}

stop() {
        echo -n "Stopping $SERVICE_NAME... "

        cmd="cd $RUN_IN && sh run.sh --stop-daemon"

        case "$(id -un)" in
                $RUN_AS)
                        eval "$cmd"
                        ;;
                root)
                        su - $RUN_AS -c "$cmd"
                        ;;
                *)
                        echo "*** ERROR *** must be $RUN_AS or root in order to control
     this service" >&2
                        exit 1
        esac

        echo "done."
}

status() {
        echo -n "$SERVICE_NAME status: "

        while read pid; do
                if [ "$(readlink -m /proc/$pid/cwd)" = "$(readlink -m $RUN_IN)" ]; then
                        echo "started"
                        return 0
                fi
        done < <(ps ax -o 'pid cmd' | grep -P '^\s*\d+ python ./scripts/paster.py serve
    ' | awk '{print $1}')
        echo "stopped"
        return 3
}

notsupported() {
        echo "*** ERROR*** $SERVICE_NAME: operation [$1] not supported"
}

usage() {
        echo "Usage: $SERVICE_NAME start|stop|restart|status"
}


#---

case "$1" in
        start)
                start "$@"
                ;;
        stop)
                stop
                ;;
        restart|reload)
                stop
                start
                ;;
        status)
                set +e
                status
                exit $?
                ;;
        '')
                usage >&2
                exit 1
                ;;
        *)
                notsupported "$1" >&2
                usage >&2
                exit 1
                ;;
esac
```

The LSB header information contained in this script is specific to our Ubuntu system's startup scripts. It allows the `update-rc.d` script to configure the startup and shutdown of Galaxy for the appropriate runlevels during system initialization and shut down.

5 Account creation and initial login

Once Galaxy has been successfully installed and brought online, we can then create our user. This section briefly covers the GUI interface, as most of the functionality is documented in the Galaxy user documentation. At this point the Galaxy instance should be running according to the previously outlined configuration and accessible via a web browser. By default, Galaxy makes use of port `8080`. Therefore, in our case, to access Galaxy we simply direct our browser to the following URL http://chromosome.med.harvard.edu:8080

From here, you will want to create the administrative account that matches the email address you specified in your configuration in the `admin_users` section of `universe_wsgi.ini`.



This Galaxy instance has been configured such that only users who are logged in may use it. If you don't already have an account, you may create one.

Login

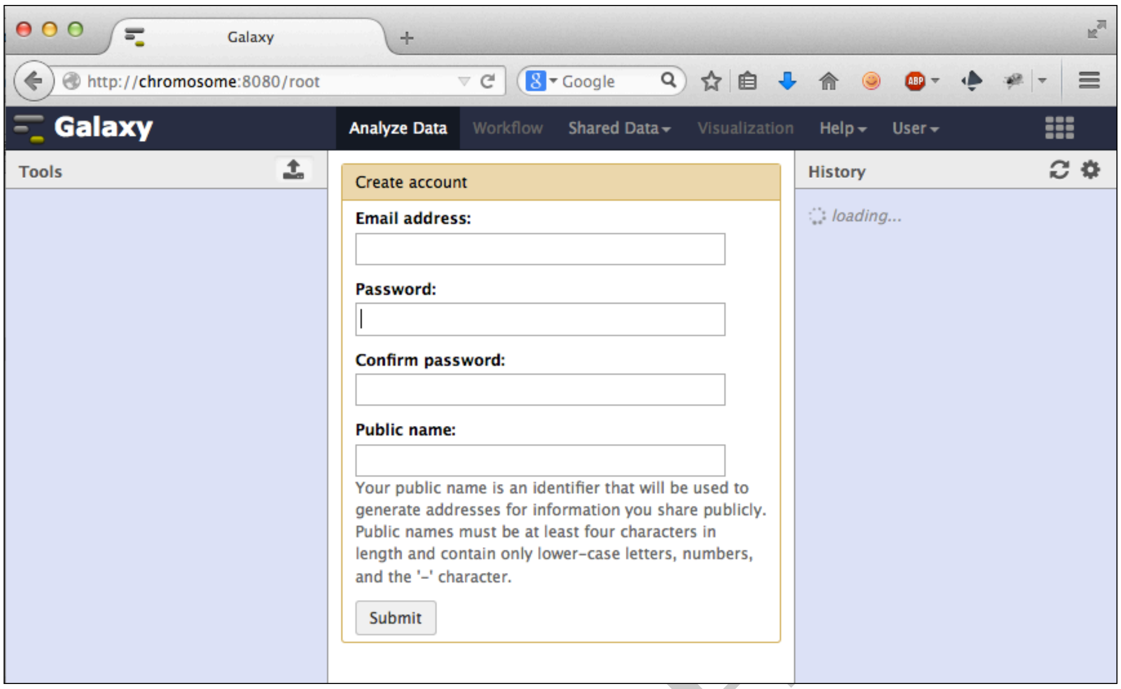
Email address:

Password:

Forgot password? Reset here

Login

From here you be greeted with a registration page that will allow you to create your account.



Create account

Email address:

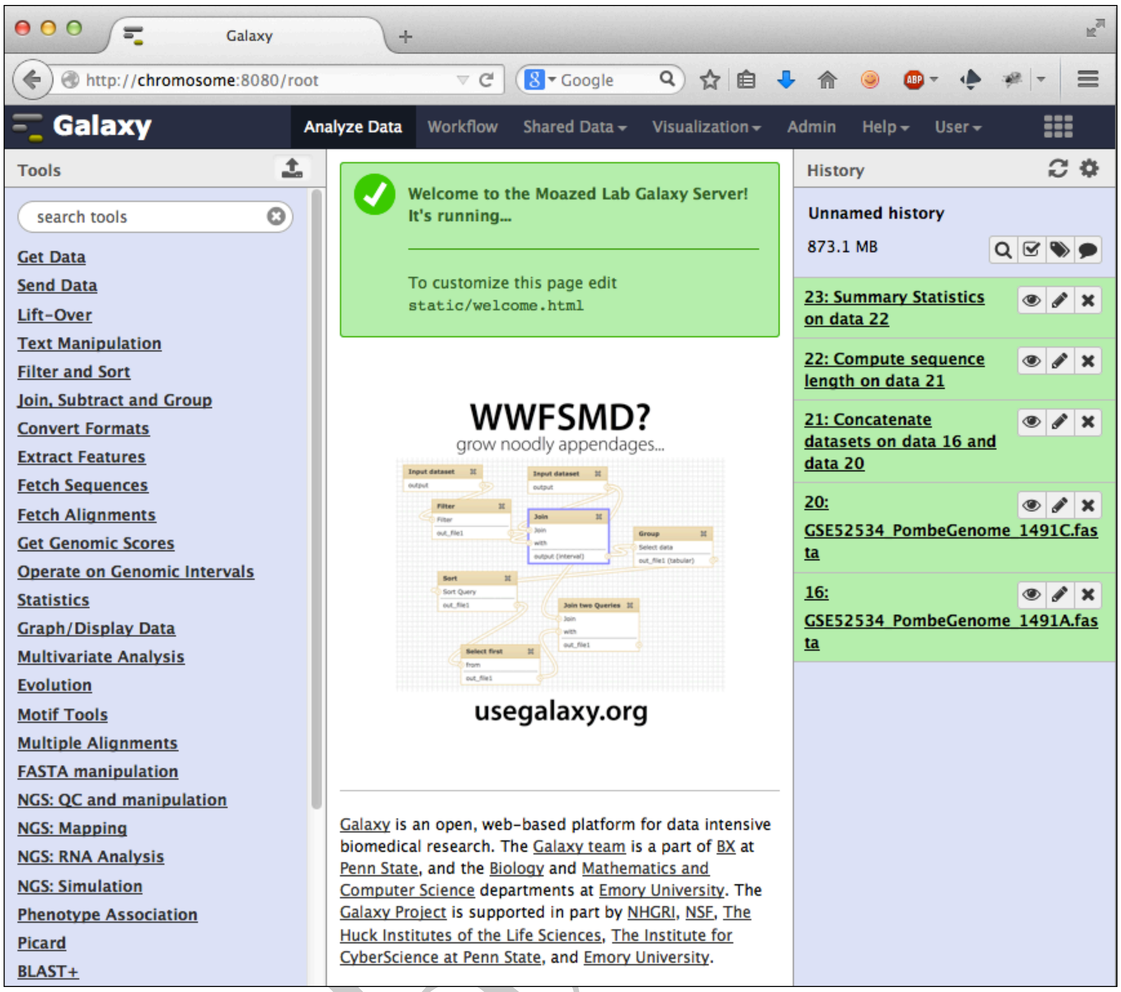
Password:

Confirm password:

Public name:

Your public name is an identifier that will be used to generate addresses for information you share publicly. Public names must be at least four characters in length and contain only lower-case letters, numbers, and the '-' character.

Once you have successfully logged in, you will be greeted with the standard Galaxy page. Please note that you should have an `Admin` menu visible for your user. (You may need to restart your Galaxy instance for this to take effect.)



Welcome to the Moazed Lab Galaxy Server!
It's running...

To customize this page edit `static/welcome.html`

Galaxy is an open, web-based platform for data intensive biomedical research. The Galaxy team is a part of BX at Penn State, and the Biology and Mathematics and Computer Science departments at Emory University. The Galaxy Project is supported in part by NHGRI, NSF, The Huck Institutes of the Life Sciences, The Institute for CyberScience at Penn State, and Emory University.

23: Summary Statistics on data 22

22: Compute sequence length on data 21

21: Concatenate datasets on data 16 and data 20

20: GSE52534_PombeGenome_1491C.fasta

16: GSE52534_PombeGenome_1491A.fasta

6 Setting up a Data Library in Galaxy

Galaxy allows for the configuration of data libraries in order to share data files. The benefit of a data library is that access control may be permissioned on a user or group basis, if necessary. Furthermore, data stored in a data library is only stored once within the database. When users import data from a data library, a pointer to the original data is passed. This reduces disk utilization on the server and circumvents duplication of large data sets shared among multiple users. We chose to implement a Data Library for the shared publication data that has been generated by researcher's within our lab. The benefits of this are several-fold. We have a central repository within the laboratory for all of our sequencing data. New lab members can easily examine work that has been done to date and thereby reduce the time required for them to acclimate themselves to the work flow within the lab. Researcher's unfamiliar or just starting out utilizing bioinformatics tools are presented with a relatively intuitive GUI interface that can assist them in performing analyses on our lab's server. The centralized repository of sequence data maintains annotation of the relevant NCBI GEO and SRA accession identifiers. In order to import the SRA data in the proper file formats applicable to Galaxy, the `sra-toolkit` was used to convert SRA formatted data to SAM file format and then uploaded to Galaxy via the Data Library management interface. The directory hierarchy for SRA data is as follows:

```
MoazedLabData
├─Processed
└─SRAs
   ├─SRP...
   └─...
```

`MoazedLabData` resides under the base `galaxy-dist`. SRAs are downloaded from NCBI and then processed via `sra-toolkit` for conversion to a bzip2-compressed SAM format (`SRRnnnnn.sam.bz2`). In order to automate the conversion of SRA data to SAM format, some helper scripts were written in `bash` and are included below.

process.sh

```bash
#!/bin/bash
for i in *.sra; do
sra2sam.sh $(echo $i | cut -d. -f1)
done
```

sra2sam.sh

```bash
#!/bin/bash
export PATH=/usr/local/sratoolkit.2.3.5-2-centos_linux64/bin:$PATH
if [[ $# != 1 ]]; then
    echo "usage $0 sra_accession"
else
    if [[ ! -a ${1}.sra ]]; then
        echo "$1 does not exist."
        exit 1
    fi
    inputfile="$1"
    outputfile="${1}.sam.bz2"
    outputdir="/usr/local/galaxy-dist/MoazedLabData/Processed"
    sam-dump -r --bzip2 --output-file ${outputdir}/${outputfile} ${inputfile}
fi
exit 0
```

Once files are downloaded from NCBI, `process.sh` is called from within the directory containing the `sra` data files, this script passes each file to `sra2sam.sh` which calls `sam-dump` (part of the sra-toolkit) to output a compressed SAM file to the `Processed` directory.

These files are then uploaded into Galaxy via the Data Library Management interface within the Admin menu of Galaxy. Once the data has been uploaded into Galaxy, is it now stored within our postgresql database.

7 Bioinformatics tools and integration with the Galaxy tool shed

There are numerous bioinformatics software tools available for use on Linux-based computing platforms. Galaxy leverages these by providing python wrappers that will present form based entry for the various parameters associated with each tool. Fortunately, the majority of these software packages already have pre-defined wrappers that can be accessed through the Galaxy tool shed. Administration of the tool shed allows us as the administrator to install these tools in Galaxy and make them available to users. It should be noted however, that the software tools themselves must also be installed and available on the Galaxy user's `$PATH` in order to be executed by Galaxy. Each software package will have its own requirements regarding acceptable version number and individual configuration. For alignment and NGS mapping tools like `bwa`, `tophat`, `tophat2`, and `BLAST`, the local indexes and libraries can be maintained on the local installation server where Galaxy is installed. A good reference regarding how to configure local indexes is available in the Galaxy documentation regarding data preparation.

For the purposes of our laboratory environment and research regarding our model ogranism, *Schizosaccaromyces pombe*, there is no available UCSC database entry regarding our model organism. Therefore, support for adding our model organism's reference genome into these built in indexes is not available. As such, a reference genome in `fasta` format was installed into a shared Data Library for use in our computational work flows.

8 Galaxy work flow creation

Work flows are created either by extracting a work flow from an existing analysis, or by building a work flow within the work flow editor. Work flows allow for the creation of a generalized pipeline for the processing of biological data using the many tools installed in the Galaxy tool shed. This approach utilizes a graphical flowchart detailing the inputs and outputs derived from each tool. Individual tool's outputs may be connected to the inputs of subsequent tools, forming a data pipeline. These work flows may then be saved and shared with other users of the Galaxy platform.

Work flows are modeled after data flow diagrams found in software engineering, whereby an input data set is transformed through a series of applied functions, resulting in output derived from the series of transformations applied to the data in the model. This allows for an abstract representation of the analysis to be performed on a type of data input which will result in data output of a specific format. Further complex analysis can then be derived by chaining relatively simple work flows to construct larger, more complex pipelines. Presented below is an example, illustrating the work flow creation process, utilizing the work flow editor.

8.1 Example work flow

As an example workflow, we have chosen to utilize the experimental data found in the Nature protocol "Differential gene expression analysis of RNA-seq experiments with TopHat and Cufflinks". For details on where to obtain the sample data for this protocol, please reference the original paper. In this analysis RNA-seq data from *Drosophila melanogaster* will be mapped to the reference genome using tophat. The expressed genes and transcripts will then be assembled utilizing cufflinks and cuffmerge will then merge the assemblies into a single transcript annotation. This data may then be analyzed through the use of CummeRbund to plot differentially expressed genes.

Sample data for the protocol was downloaded to the server running our Galaxy instance. This data was then uploaded into our Galaxy instance through the Data Library management interface.

8.2 A note about tophat

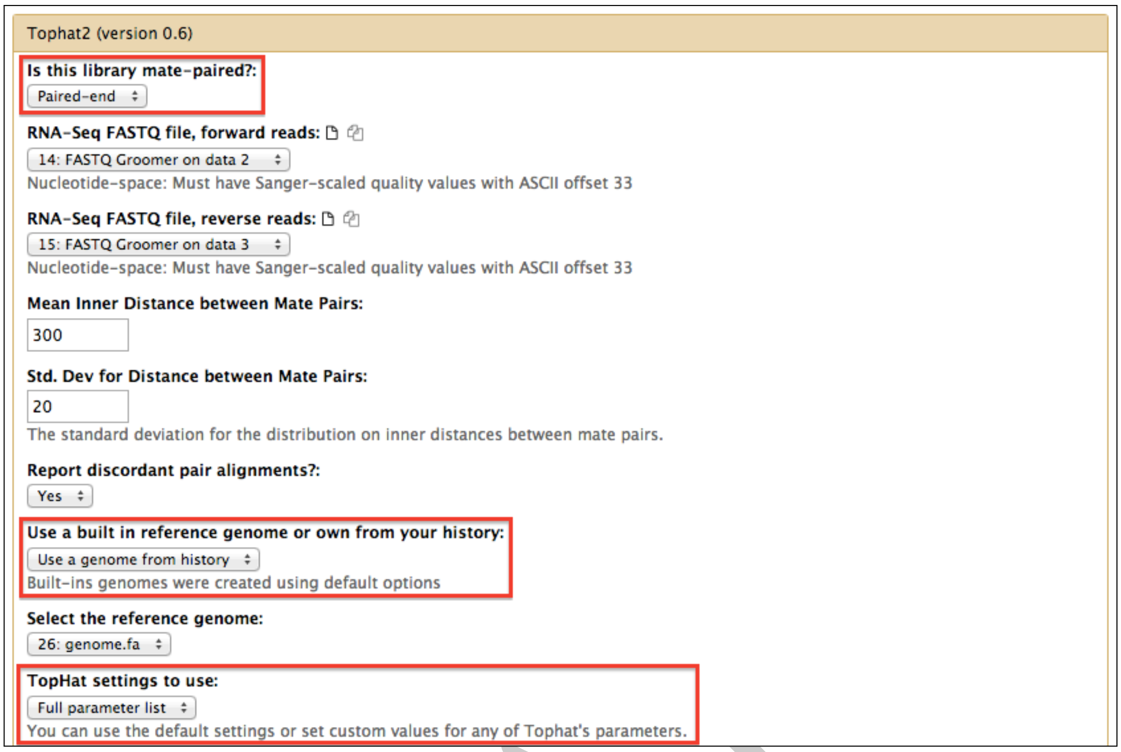
In order to run the tophat portion of the protocol, we had to apply a manual patch to fix option passing to tophat2. The details of this patch are outlined here. It is possible that this patch has already been incorporated into a newer release of the tophat software, however as of the time of this writing, we were running the most recent release of tophat.

```
root@chromosome:~# tophat --version
TopHat v2.0.11
```

While the procedure for performing a differential gene analysis follows closely the protocol outlined in the Nature paper, there are some additional steps that must be considered when performing the analysis within the Galaxy environment. The first thing to take note of is that our fastq data will need to be run through the fastqsanger groomer to convert the quality scores. Once this is done, we then may run `tophat2` to align the RNA-seq reads to the *Drosophila melanogaster* reference genome. We must also import files that we wish to use into our current history, so that they may be passed as arguments within the tool dialogs. Therefore it is important to import both the annotation file `genes.gtf` and `genome.fa` into the history, along with the RNA-seq fastq data files to be aligned.

Once the files have been imported from our Data Library, we first run the `fastqsanger groomer` on each of the RNA-seq files. We then select the `tophat2` tool and select the appropriate options to perform our alignment.

Below is a series of reference screenshot detailing the options selected for running tophat2 against the first set of paired-end reads in our RNA-seq data set.



Tophat2 (version 0.6)

Is this library mate-paired?:
Paired-end

RNA-Seq FASTQ file, forward reads:
14: FASTQ Groomer on data 2
Nucleotide-space: Must have Sanger-scaled quality values with ASCII offset 33

RNA-Seq FASTQ file, reverse reads:
15: FASTQ Groomer on data 3
Nucleotide-space: Must have Sanger-scaled quality values with ASCII offset 33

Mean Inner Distance between Mate Pairs:
300

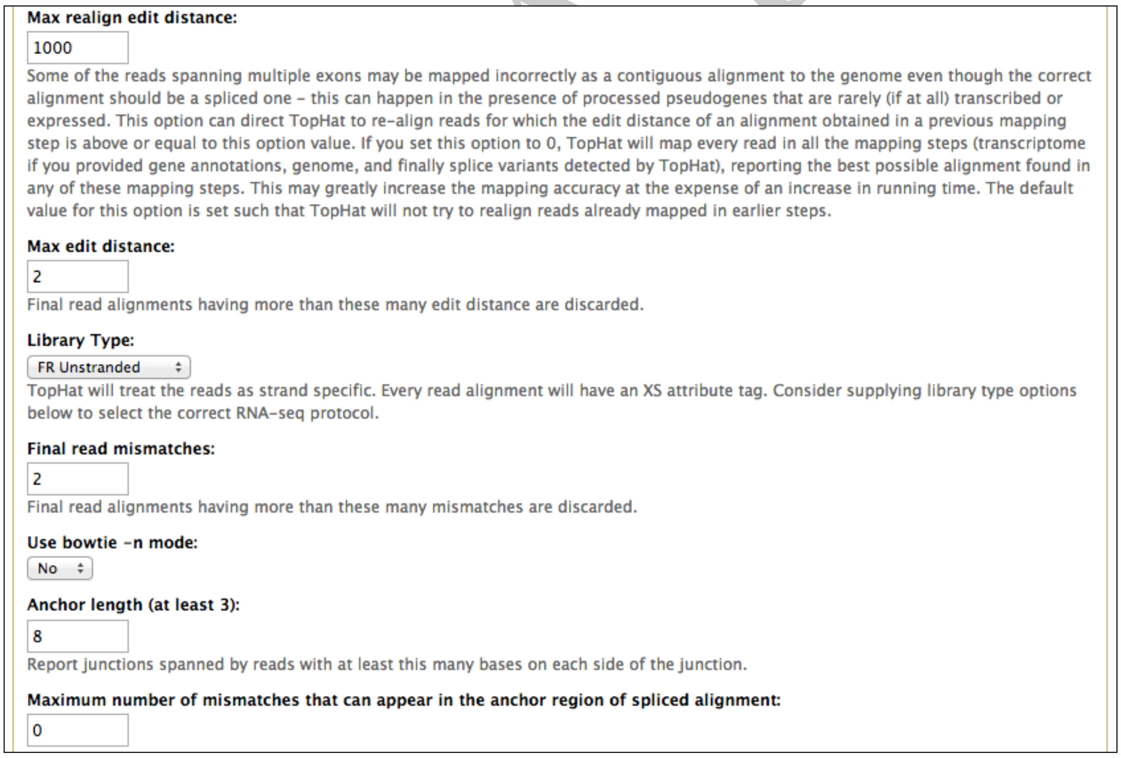
Std. Dev for Distance between Mate Pairs:
20
The standard deviation for the distribution on inner distances between mate pairs.

Report discordant pair alignments?:
Yes

Use a built in reference genome or own from your history:
Use a genome from history
Built-ins genomes were created using default options

Select the reference genome:
26: genome.fa

TopHat settings to use:
Full parameter list
You can use the default settings or set custom values for any of Tophat's parameters.



Max realign edit distance:
1000
Some of the reads spanning multiple exons may be mapped incorrectly as a contiguous alignment to the genome even though the correct alignment should be a spliced one - this can happen in the presence of processed pseudogenes that are rarely (if at all) transcribed or expressed. This option can direct TopHat to re-align reads for which the edit distance of an alignment obtained in a previous mapping step is above or equal to this option value. If you set this option to 0, TopHat will map every read in all the mapping steps (transcriptome if you provided gene annotations, genome, and finally splice variants detected by TopHat), reporting the best possible alignment found in any of these mapping steps. This may greatly increase the mapping accuracy at the expense of an increase in running time. The default value for this option is set such that TopHat will not try to realign reads already mapped in earlier steps.

Max edit distance:
2
Final read alignments having more than these many edit distance are discarded.

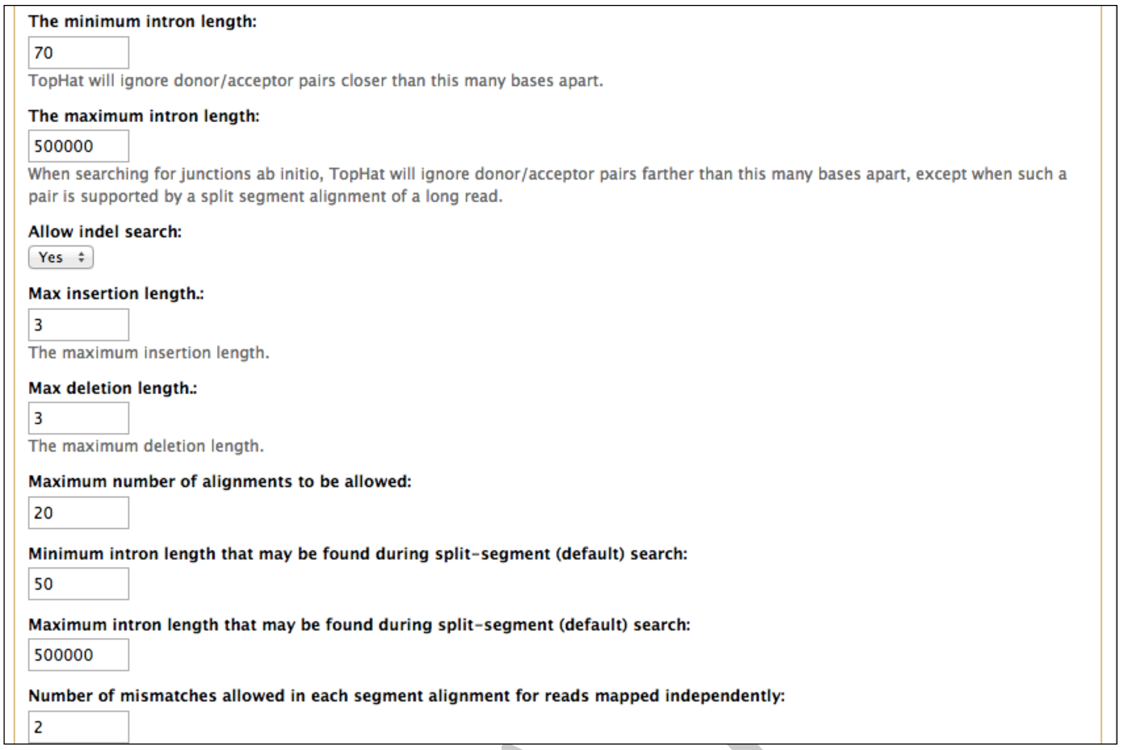
Library Type:
FR Unstranded
TopHat will treat the reads as strand specific. Every read alignment will have an XS attribute tag. Consider supplying library type options below to select the correct RNA-seq protocol.

Final read mismatches:
2
Final read alignments having more than these many mismatches are discarded.

Use bowtie -n mode:
No

Anchor length (at least 3):
8
Report junctions spanned by reads with at least this many bases on each side of the junction.

Maximum number of mismatches that can appear in the anchor region of spliced alignment:
0



The minimum intron length:

70

TopHat will ignore donor/acceptor pairs closer than this many bases apart.

The maximum intron length:

500000

When searching for junctions ab initio, TopHat will ignore donor/acceptor pairs farther than this many bases apart, except when such a pair is supported by a split segment alignment of a long read.

Allow indel search:

Yes

Max insertion length.:

3

The maximum insertion length.

Max deletion length.:

3

The maximum deletion length.

Maximum number of alignments to be allowed:

20

Minimum intron length that may be found during split-segment (default) search:

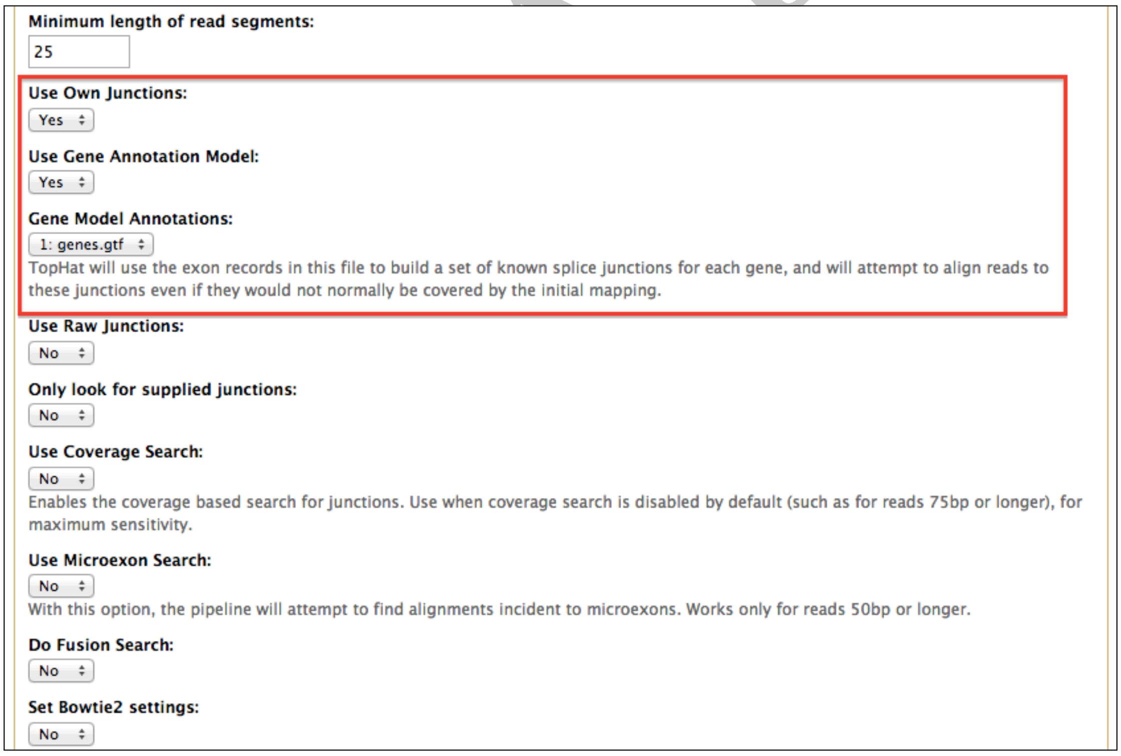
50

Maximum intron length that may be found during split-segment (default) search:

500000

Number of mismatches allowed in each segment alignment for reads mapped independently:

2



Minimum length of read segments:

25

Use Own Junctions:

Yes

Use Gene Annotation Model:

Yes

Gene Model Annotations:

1: genes.gtf

TopHat will use the exon records in this file to build a set of known splice junctions for each gene, and will attempt to align reads to these junctions even if they would not normally be covered by the initial mapping.

Use Raw Junctions:

No

Only look for supplied junctions:

No

Use Coverage Search:

No

Enables the coverage based search for junctions. Use when coverage search is disabled by default (such as for reads 75bp or longer), for maximum sensitivity.

Use Microexon Search:

No

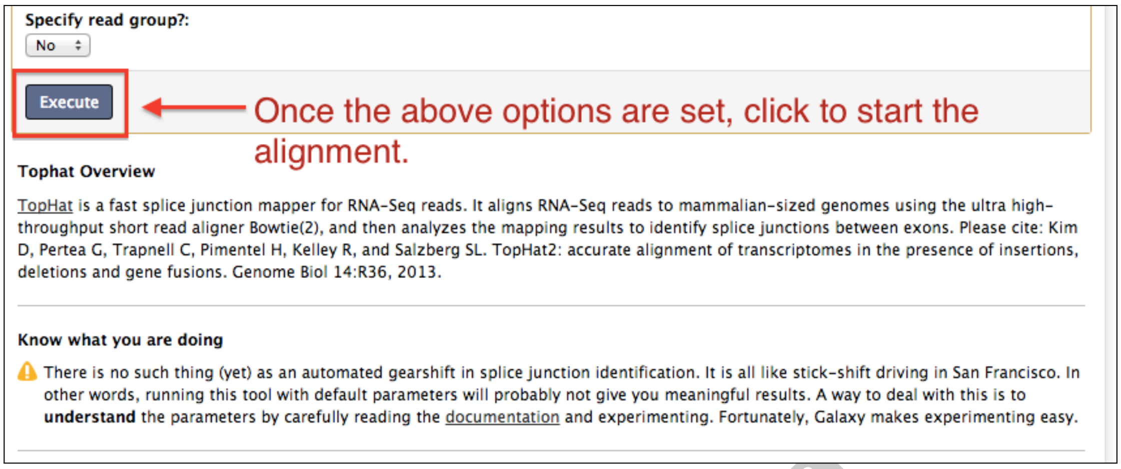
With this option, the pipeline will attempt to find alignments incident to microexons. Works only for reads 50bp or longer.

Do Fusion Search:

No

Set Bowtie2 settings:

No


Specify read group?:
No

Execute

Once the above options are set, click to start the alignment.

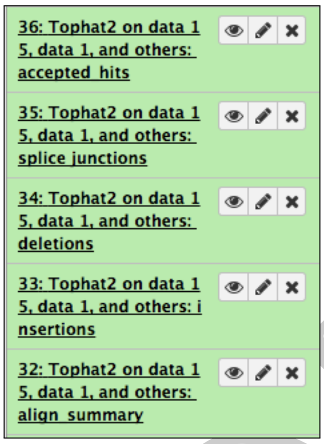
Tophat Overview

TopHat is a fast splice junction mapper for RNA-Seq reads. It aligns RNA-Seq reads to mammalian-sized genomes using the ultra high-throughput short read aligner Bowtie(2), and then analyzes the mapping results to identify splice junctions between exons. Please cite: Kim D, Pertea G, Trapnell C, Pimentel H, Kelley R, and Salzberg SL. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. Genome Biol 14:R36, 2013.

Know what you are doing

There is no such thing (yet) as an automated gearshift in splice junction identification. It is all like stick-shift driving in San Francisco. In other words, running this tool with default parameters will probably not give you meaningful results. A way to deal with this is to **understand** the parameters by carefully reading the documentation and experimenting. Fortunately, Galaxy makes experimenting easy.

Once the run has completed, the history sidebar will be updated. provided the run was successful and contained no errors during execution, you should see something similar to this in your history.


36: Tophat2 on data 15, data 1, and others: accepted_hits

35: Tophat2 on data 15, data 1, and others: splice junctions

34: Tophat2 on data 15, data 1, and others: deletions

33: Tophat2 on data 15, data 1, and others: insertions

32: Tophat2 on data 15, data 1, and others: align_summary

Clicking on the eye icon of the `align summary` will display a summary of the alignment in the main window.

```
Left reads:
          Input     :  11607353
           Mapped   :  11607353 (100.0% of input)
            of these:     61370 (  0.5%) have multiple alignments (159 have >20)
Right reads:
          Input     :  11607353
           Mapped   :  11607352 (100.0% of input)
            of these:     61370 (  0.5%) have multiple alignments (159 have >20)
100.0% overall read mapping rate.

Aligned pairs:  11607352
     of these:     61370 (  0.5%) have multiple alignments
                      28 (  0.0%) are discordant alignments
100.0% concordant pair alignment rate.
```

9 Conclusions