B. Managing Bioinformatics Tools

16. Installing Bioinformatics Software in a Server-Based Bioinformatics Resource

17. Management of a Server-Based Bioinformatics Resource

Job:
Chapter: 16/Fris
Pub Date:
Template:Krawitz7x10

Operator: NF
Date: 7.25.02
Revision: Corrections/Paging

16 Installing Bioinformatics Software in a Server-Based Computing Environment

Brian Fristensky

Introduction

To support a diverse institutional program of genomics projects, it is often necessary to have an equally diverse and comprehensive software base. Although programs may come from many sources, it is important to make them easily accessible to the user community on a single computing platform. This chapter will outline the strategies for installing programs for a server-based molecular biology software resource, accessed by a large user base. It is assumed that the reader is familiar with basic UNIX commands and concepts, as described in the previous chapters. The approaches discussed here are implemented in the BIRCH system (*see* Website: http://home.cc.umanitoba.ca/~psgendb), but are generally applicable to any centralized multiuser software installation. The important parts of the process are described in either program documentation or UNIX documentation. The tricks and conventions that help to simplify the installation process will also be highlighted. This should give the novice an idea of what to expect before wading into the documentation.

Considerations

There are five guiding principles for installation and use that should be applied to help ensure a smooth operation.

1. Any user should be able to run any program from any directory simply by typing the name of the program and arguments. It should not be necessary to go to a specific directory to run a program.
2. System administration should be kept as simple as possible. This saves work for the Bioadmin[1], as well as increasing the likelihood that things will function properly.
3. Avoid interruption of service during installation and testing.
4. The Bioadmin should never have to modify individual user accounts.
5. Even if you have root access, do most of your work on a regular user account. Log in as root only when necessary.

[1]Since bioinformatics software may be installed by a specialist other than UNIX system staff, the term *Bioadmin* will refer to the person installing and maintaining bioinformatics software, distinct from system administrators.

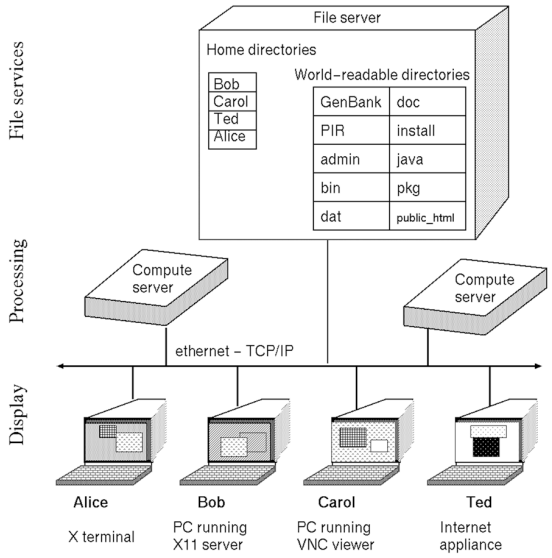

Fig. 1. Network-centric computing.

The Networked Computing Environment

The casual computer user learns only a narrow computing model: the PC model. PCs are based on the idea that each person has their own computer that is completely self-contained, with all hardware, software, and data residing physically on the desktop. Provisions for multiple users on a single machine (e.g., separate home directories, user accounts, file permissions) may exist, but are seldom taken into account by PC software developers. Each PC becomes a special case with special problems. The work of administration grows with the number of computers. Software has to be purchased and installed independently for each machine. Security and backup are often not practiced.

See companion CD for color Fig. 1

UNIX greatly simplifies the problem of computing with a network-centric approach, in which any user can do any task from anywhere. Figure 1 illustrates computing in a network-centric environment. All data and software reside on a file server, which is remotely mounted to one or more identically configured computer servers. Programs are executed on a computer server, but displayed at the user's terminal or PC. Regardless of whether one logs in from an X11 terminal, a PC running an X11 server, a PC using the VNC viewer (*see* Chapters 13 and 17) or an internet appliance, the user's desktop screen looks the same and opens to the user's $HOME directory. Consequently, any user can do any task from any device, anywhere on the Internet.

Leveraging the Multi-Window Desktop

The installation process involves moving back and forth among several directories, which is most effectively accomplished by viewing each directory in a separate window. One of the things that makes the typical PC desktop awkward to use is the *one window owns the screen* model. In MS-Windows, most applications default to

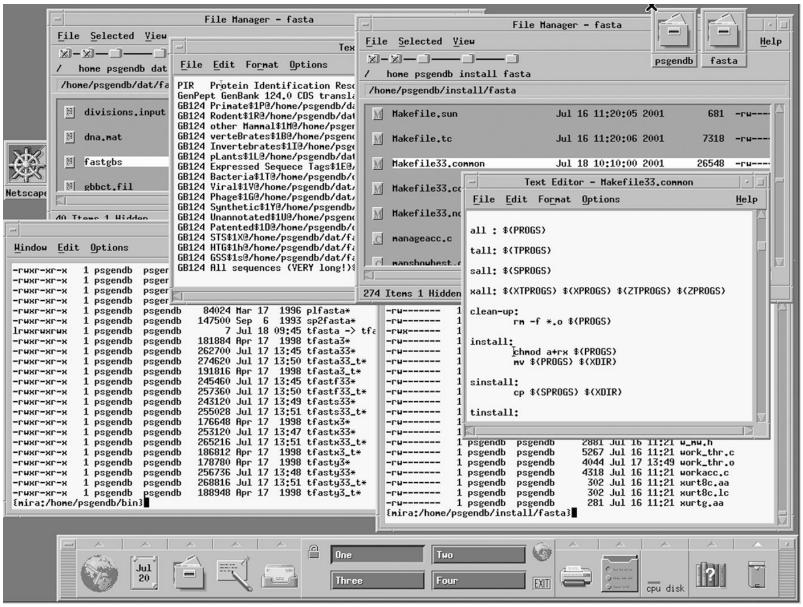

Fig. 2. Leveraging the multiwindow desktop. In the example, the environment variable $DB, which identifies the root directory for bioinformatics software, is set to /home/psgendb. Clockwise from top left: Double clicking on **fastgbs** in $DB/doc/fasta opens up a list of locations of database files in a text editor. Similarly, Makefile33.common has been opened up from $DB/install/fasta. This file contains Makefile commands that are compatible with all operating systems. A terminal window at lower right is used for running commands in $DB/install/fasta, while another terminal window at bottom left is used for running commands in $DB/bin. At bottom, the CDE control panel shows that the current screen, out of four screens available, is screen One.

Note: To get the C-shell to display the hostname and current working directory in the prompt, include the following lines in cshrc.source or your .cshrc file:

```
set HOSTNAME = `hostname`
set prompt="{$HOSTNAME":"$cwd"}
alias cd `cd \!*; set prompt="{$HOSTNAME":"$cwd"}'
alias popd `popd \!*; set prompt="{$HOSTNAME":"$cwd"}'
alias pushd `pushd \!*; set prompt="{$HOSTNAME":"$cwd"}'
```

take up the entire screen. One moves between applications using the task bar. Although it is possible to resize windows so that many can fit on one screen, this is seldom done in MS-Windows. On Macintosh, even when multiple windows are present, they depend on the menu at the top of the screen, requiring the user to first select a window by clicking on it, and then to choose an item from the menu at the top of the screen. Even worse, the menus often look almost identical from program to program, so that it is not obvious when the focus has shifted to a new application.

On UNIX desktops, menus are found within the windows themselves. This decreases the amount of distance the eye has to cover. Focus moves with the mouse and does not need to be switched with the taskbar. The user simply moves from one window to another and works. Because UNIX tends to be oriented towards multiple windows, UNIX users tend to favor larger monitors. More screen real estate means more space for windows. The screen in Fig. 2 appears crowded because it was gener-

See companion CD for color Fig. 2

Table 1
Sources of Free Downloadable Software

Source	*URL*
IUBio Archive	http://iubio.bio.indiana.edu/
EMBOSS Software Suite	http://www.uk.embnet.org/Software/EMBOSS/
Open Source Bioinformatics Software	http://bioinformatics.org/
Linux for Biotechnology	http://www.randomfactory.com/lfb/lfb.html
Sanger Center Software	http://www.sanger.ac.uk/Software/
Staden Package	http://www.mrc-lmb.cam.ac.uk/pubseq/
NCBI FTP site	http://www.ncbi.nlm.nih.gov/Ftp/index.html
PHYLIP Phylogeny software	http://evolution.genetics.washington.edu/phylip.html
BIRCH, FSAP, XYLEM,GDE	http://home.cc.umanitoba.ca/~psgendb/downloads.html
FASTA package	ftp://ftp.uva.edu/pub/fasta/
Virtual Network Computing (VNC)	http://www.uk.research.att.com/vnc/

ated at 1024 × 768 resolution. Although this is a common resolution for PCs, the UNIX community tends to work with larger monitors, at 19" diagonal or larger, running at 1280 × 1000 or higher resolution. To provide further real estate, UNIX desktops such as CDE, KDE, and GNOME support switching between several desktops at the push of a button. Use of multiple windows during an installation is illustrated in Figure 2.

Finding and Downloading Software

Table 1 has a short and by no means exhaustive list of sites where freely available sequence analysis software can be downloaded. USENET newsgroups such as bionet.software contain announcements of new software and updates, as well as discussions on molecular biology software.

Usually, software is downloaded as a directory tree packed into a single archive file in various formats. Generally, files in these formats can recreate the original directory tree containing source code, documentation, data files, and often, executable binaries. Usually, the first step is to uncompress the file and then recreate the original directory. For example, the fasta package comes as a shell archive created using the shar command. Because you do not know in advance whether the archive contains a large number of individual files or a single directory containing files, it is always safest to make a new directory in which to recreate the archive, using the following commands:

mkdir fasta	create new directory
mv fasta.shar fasta	move fasta.shar into the new directory
cd fasta	go into the fasta directory
unshar fasta.shar	extract files from fasta.shar

Table 2 lists some of the most common archive tools and their usage.

Two goals when installing software are to 1) avoid interruption of service for users during installation and testing and 2) having the option of deleting programs after

Table 2
Archive Commands

File extension	*Utility*	*Archive command*	*Unarchive command*
.tar	UNIX tar	To create a tar file from a directory called source: **tar cvf source.tar source**	To recreate the directory: **tar xvf source.tar**
.zip	ZIP	To create a compressed archive file called source.zip: **zip source source**	To recreate the directory: **unzip source**
.shar	UNIX shar	To create a shar file from a directory named source: **shar source > source.shar**	To recreate the directory: **unshar source.shar** or **chmod u+x source.shar** or **sh source.shar**[a]
.gz	GNU zip	To compress a file with: **gzip source > source.gz**	To recreate the directory: **gunzip source**
.Z	compress	To compress source.tar: **compress source.tar**	To uncompress source.tar.Z: **uncompress source.tar.Z**
.uue	uuencode	To encode source.tar.Z using ASCII characters: **uuencode source.tar.Z source.uue**	To recreate the original binary file: **uudecode source.uue**

[a].shar files are actually shell scripts that can be executed to recreate the original directory.

evaluation. For example, a separate directory called *install* could hold separate directories for each package during the installation.

Understand the Problem Before You Begin

For many standard office tasks, it is possible to get by without ever reading the documentation. In molecular biology, the task itself often has enough complexity that it may not be possible to simply launch, point, and click. In practice, it is almost always faster to read the documentation before trying to install. In particular, each program will have installation instructions that let you know about important options for where the final program files will reside and which environment variables must be set.

Reading the documentation at this stage gives you a chance to learn more about what the program does and to decide if it is really what you need. This weeding out phase can save a lot of unnecessary compiling, organizing, and testing.

Compilation

Programs distributed as source code, for which no binaries are available, will require compilation and linking steps. Although these procedures vary somewhat with language, most of the common packages use protocols of the C and C++ family of

Table 3
Common File Types and File Extensions

File extension	*File type*
.c	C source code
.h	C header
.o	Compiled object file
no extension	Executable binary file
.1, .l	UNIX manual page
.makefile, .mak	Makefile

languages. In addition to source files (.c), code items such as type definitions, which need to be shared, are found in header files (.h). At compile time, code from header files is inserted into C code, and the .c file translated to machine code, which is written as object modules (.o). Next, the compiler calls a linker, which links object modules into a final executable file. In most programs, object modules from standard libraries (e.g., Tcl/Tk) are also linked. These are typically linked dynamically, meaning that only a reference to the libraries is made and the actual library modules are loaded each time the program is run. Consequently, dynamic linking saves disk space. However, when a program depends on libraries that may not be found on all systems, static linking can be accomplished, in which object modules are written to the final executable code file. Static linking favors portability at the expense of disk space. A short list of the types of files frequently encountered during installation appears in Table 3.

Virtually all scientific program packages automate these procedures using the *make* program. The *make* program reads a Makefile, containing compilation, linking, and installation options. For cross-platform compatibility, it is common to include separate Makefiles for each platform (e.g., SGI, Linux, Windows, Solaris). For example, the fasta package has a file called Makefile.sun for Solaris systems. Copy Makefile.sun to Makefile, and edit Makefile as needed for your system. At the beginning of the Makefile, variables are often set to specify the final destinations for files. On our system, fasta's Makefile would be edited to change the line reading **XDIR = /seqprog/sbin/bin to XDIR = /home/psgendb/bin.** Because this directory is in the $PATH for all BIRCH users, the new programs become available to all users as soon as the files are copied to this location.

Typing **make** executes the commands in Makefile, compiling and linking the programs. It is best to run make in a terminal window that supports scrolling, so that all warning and error messages can be examined. This is particularly important because one can then copy error messages to a file to provide the author of the program with a precise description of the problem. If the authors do not receive this feedback, the problems do not get fixed. However feedback must be precise and detailed.

If make is successful, executable binary files, usually with no extension, are written to the target directory, which may or may not be the current working directory. Many Makefiles require you to explicitly ask for files to be copied to the destination directory by typing **make install**.

In some cases, testing can be carried out at this point, particularly if a test script is included with the package. In the fasta package several test scripts are found. For example, **./test.sh** will run most of the fasta programs with test datafiles.

Note: "./" forces the shell to look for test.sh in the current working directory. Unless "." was in your $PATH, it would not be found. It is generally considered insecure to include "." in your $PATH.

It is important to check test scripts to see where they look for executable files. For example, if the script sets $PATH to "." (the current directory), or if programs are executed with a statement such as **./fasta33**, then the shell will look for an executable file in the current directory. If the directory for the executable file is not explicitly, set, the shell will search all directories in your $PATH to find an executable file. This could either result in a *Command not found* message, or if earlier copies of the programs were already installed, these older programs execute, not the newly compiled programs.

Installation

In the BIRCH system, all files and directories are found in a world-readable directory specified by the $DB environment variable. Thus, $DB/bin, $DB/doc, and $DB/dat refer to directories containing executable binaries, documentation, and datafiles used by programs, respectively, as summarized below.

$DB/bin

Although $DB/bin could in principle be set to refer to /usr/local/bin, it is probably best to keep the entire $DB structure separate from the rest of the system. This approach has the advantage that the Bioadmin need not have root privileges. All files in $DB/bin should be world-executable.

One practice for managing program upgrades is to create a symbolic link to point to the current *production* version of the program. For example, a link with the name *fasta* might point to fasta3:

```
lrwxrwxrwx 1 psgendb psgendb 6 Jul 18 09:45 /home/psgendb/bin/fasta3 -> fasta*
```

To upgrade to fasta33:

```
rm fasta
ln -s fasta33 fasta
lrwxrwxrwx 1 psgendb psgendb 6 Jul 18 09:45 /home/psgendb/bin/fasta33 -> fasta*
```

Aside from giving users a consistent name for the current most recent version of the program, this type of stable link eliminates the need to modify other programs that call the upgraded program.

$DB/doc

Documentation files for software should be moved to this directory. Ideally, the complete contents of this directory should be Web-accessible. Where a program or package has more than one documentation file, create a separate subdirectory for each program. All files should be world-readable.

$DB/dat

A program should never require that ancillary files such as fonts or scoring matrices be in a user's directory. These should always be centrally installed and administered, transparently to the user. Datafiles required for programs, such as scoring

matrices, lists of restriction enzymes, and so forth should be moved to this directory. Generally, each program should have its own subdirectory. All files should be world-readable.

$DB/admin

This directory contains scripts and other files related to software administration. First time BIRCH users run the *newuser* script, to append a line to the user's .login file:

source /home/psgendb/admin/login.source

and to the .cshrc file:

source /home/psgendb/admin/cshrc.source

These files respectively, contain commands that are executed when the user first logs in, and each time a program is started. All environment variables, aliases, and other settings required to run these programs are set in these files. Having run *newuser* once, a user should never have to do any setup tasks to be able to run new or updated programs. When a new program is added, the environment variables and aliases needed are specified in cshrc.source, and therefore become immediately available to the user community. The net effect is that the Bioadmin should never have to go to each user's account when a new program is installed.

Where programs require first-time setup, such as a configuration file being written to the user's $HOME directory, the program should be run from a wrapper script that checks for the presence of that file. If the file is not present, the script writes a default copy of the file to $HOME. The user should never have to explicitly run a setup script before using a program.

All directories that are to be accessible to users must be world searchable (world-executable), as well as world-readable. For example, to allow users to read files in $DB/doc/fasta, both $DB/doc and $DB/doc/fasta must be world executable:

chmod a+rx $DB/doc/fasta

Special Considerations for Complex Packages

Some packages come as integrated units whose components can not be moved out of their directory structure to $DB/bin/, $DB/dat, and $DB/doc. Packages of this type are installed in $DB/pkg. A good case in point is the Staden Package (*see* Table 1). The BIRCH login.source file contains the following lines:

```
# Environment variables for Staden
setenv STADENROOT $DB/pkg/staden
source $STADENROOT/staden.login
```

that cause the commands in staden.login to be executed when the user logs in. This script in turn sets several environment variables referencing files in the $STADENROOT directory. As well, login.source adds $DB/pkg/staden/solaris-bin to the $PATH. This illustrates that there are sometimes no simple solutions. On one hand, it would be desirable to simply copy all Staden binaries into $DB/bin, which has the effect of making it difficult to identify the origin of specific programs in

$DB/bin as coming from the Staden Package. On the other hand, a compromise might be to make symbolic links from $DB/bin to each of the programs in $DB/pkg/staden/solaris-bin, which would require that links be individually maintained.

Documentation and data can be linked more easily in $DB/doc. The following

ln -s $STADENROOT/doc staden

creates a link to the Staden documentation directory, while in $DB/dat, and

ln -s $STADENROOT staden

creates a link to the main Staden directory, from which the user can find several directories with sample datafiles. In this fashion, the $DOC and $DAT directories appear to contain *staden* subdirectories, whose contents are physically located elsewhere.

On Linux systems, complex packages are maintained using programs such as Red Hat Package Manager (RPM). RPM automates package installation, often requiring no user input. As files are copied to their destinations, their locations are recorded in the RPM database (/usr/lib/rpm). In addition to installation, RPM automates package updating, verification and de-installation. Tools such as RPM therefore make it possible to install software in system directories such as /usr/local/bin or /usr/bin without making these directories unmanageable. The one disadvantage is that installation in system directories can only be accomplished with root permissions.

Special Considerations for Java Applications

Java applications should be installed in a central location, such as $DB/java. Ideally, all that should be required is the inclusion of $DB/java in the $CLASSPATH environment variable. The Java Virtual Machine (JVM) would search this location at runtime. However, the precise commands needed to launch an application vary so that no single solution exists. For example, some applications are completely contained in a single .jar file, while others require a complex directory structure with large numbers of objects and datafiles. Consequently, Java applications should be launched from wrappers: short scripts that reside in $DB/bin and call the application. For example, a script called $DB/bin/readseq runs the Java implementation of readseq (available from IUBio, *see* Table 1):

```
#!/bin/csh
# UNIX script file to run command line readseq

# Full path must be specified to enable us to
# launch readseq from any directory.
setenv CLASSPATH $DB/java/readseq/readseq.jar

# $argv passes command line arguments from the wrapper
# to the application
java -cp $CLASSPATH run $argv
```

Thus, typing **readseq** launches the wrapper, which in turn launches the Java readseq application. Readseq also has a method called *app* which runs readseq in a graphic interface. To run in this mode, the Xreadseq wrapper has the following line:

java -cp $CLASSPATH app

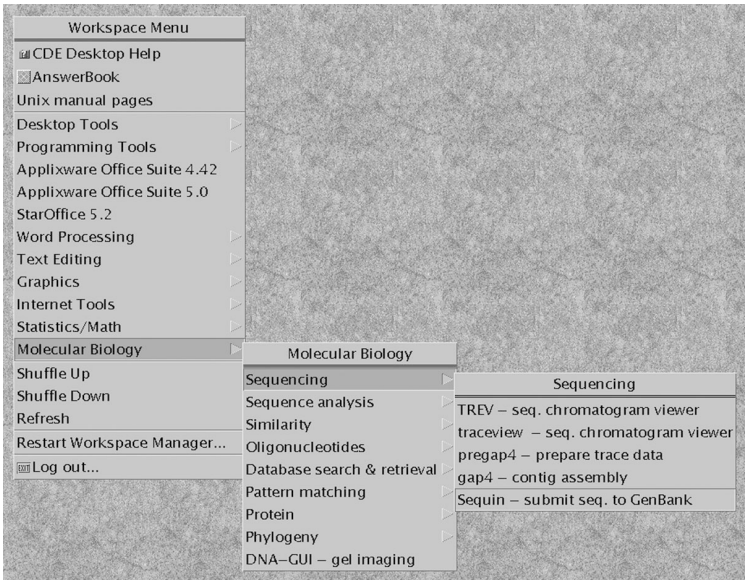

Fig. 3. The CDE Workspace Menu.

Calling Programs from a Graphic Front-End

Several options exist for unifying a large software base with a graphic front end. Programs such as GDE (*see* Table 1), SeqLab from the GCG package (*see* Website: http://www.accelrys.com), or SeqPup (*see* Table 1, IUBio) allow the Bioadmin to add external programs to their menus, as specified in easy-to-edit configuration files. In general, the user selects one or more sequences to work with, and then chooses a program from the main menu. A window pops up, allowing parameters to be set. The front end then generates a UNIX command to run the program with these parameters, using the selected sequences.

In many cases, it is best for the front end to call a wrapper that verifies and checks the parameters and sequences, then executes the program. This is especially important for programs from packages such as FSAP or PHYLIP (*see* Table 1), which operate through text-based interactive menus. If a prompt does not receive a valid response, programs of this type may go into an infinite loop, prompting for a response.

Launching Programs from the Workspace Menu

See companion CD for color Fig. 3

The *workspace* menu is yet another avenue through which users can find programs. Figure 3 shows a CDE workspace menu organized categorically. At the highest level are the main categories of programs, including standalone items for office packages. The *molecular biology* menu is further divided into submenus. For example, the *Sequencing* submenu contains programs that together cover all steps in the sequencing process, including reading the raw chromatograms, vector removal, contig assembly, and submission to GenBank. The downside of the workspace menu is that it is incomplete, as command line applications, cannot be launched from the workspace menu.

One should be able to launch GUI applications from the workspace menu, defaulting to the user's $HOME directory. In the CDE desktop, this is specified in the .dt directory. Most recent UNIX desktops, such as CDE 1.4 and GNOME use tree-structured directories to define the structure of the workspace menu. Again, it is important to avoid having to do updates to individual user accounts. The easiest way is to create a directory for molecular biology programs on the Bioadmin's account, and have new users run a script creating a symbolic link from their workspace menu directory (e.g., .dt/Desktop) to the Bioadmin's directory. Subsequently, all updates to the Bioadmin's menu will become available to all users.

Testing

Testing should not be carried out using the account that owns the programs (e.g., root). Testing should always be done in a regular user account. One reason is that testing on a user account will uncover incorrect permissions. This is likely the single most common installation error. At the same time, it is probably also best to test in a subdirectory, rather than in the $HOME directory, to fully demonstrate that the program can be run from anywhere.

Using VNC (see Table 1), one can easily eliminate login/logout cycles between your Bioadmin account and your user account. The vncserver is an X11 server that runs on a networked UNIX host. It creates an X11 screen in memory. To display that screen, run vncviewer on your desktop machine. The complete UNIX desktop appears in a window. For example, the entire screen shown in Fig. 2 was run in a vncviewer window. Thus, switching back and forth between the user desktop and the Bioadmin desktop is as easy as switching between windows, facilitating rapid test-modify cycles. (See Chapter 17 for additional VNC insights.)

Installation Checklist

Before announcing updates or new programs, go through the package in a user account, checking the following:

- All files world-readable (chmod a+r *filename*)
- All binaries world-executable (chmod a+x *filename*)
- All directories world-searchable (chmod a+x *directoryname*)
- Environment variable set in cshrc.source
- Documentation and datafiles updated

Although installation should result in a *finished product*, there are often bugs that need to be worked out as the package or program gets used. At this point, the user base is probably the best group of testers, becuase they will make mistakes, and they will try a wider range of data than the Bioadmin would try.

Acknowledgments

Thanks to the Academic Computing and Networking staff at the University of Manitoba for UNIX system support. This work was made possible in part through hardware provided by the Sun Academic Equipment Grants Program.

Au: Please provide "Glossary".

Suggested Readings

Introduction

Fristensky, B. (1999) Building a multiuser sequence analysis facility using freeware, in: Bioinformatics Methods and Protocols, (Misener, S. and Krawetz, S., eds.), Humana Press, Totowa, NJ, pp. 131–145.

Sobell, M. G. (1995) A Practical Guide to the UNIX System, Addison-Wesley Publishing, Reading, MA.

<Au/Ed: City, state correct?

The Networked Computing Environment

Fristensky, B. (1999) Building a multiuser sequence analysis facility using freeware, in: Bioinformatics Methods and Protocols, (Misener, S. and Krawetz, S., eds.), Humana Press, Totowa, NJ, pp. 131–145.

Compilation

Pearson, W. R. (1999) Flexible sequence similarity searching with the FASTA3 program package, in: Bioinformatics Methods and Protocols, (Misener, S. and Krawetz, S., eds.) Humana Press, Totowa, NJ, pp. 185–219.

Installation

Fristensky, B., Lis, J. T., and Wu, R. (1982) Portable microcomputer software for nucleotide sequence analysis, Nucl. Acids Res. 10, 6451–6463.

Fristensky, B. (1986) Improving the efficiency of dot-matrix similarity searches through use of an oligomer table, Nucl. Acids Res. 14, 597–610.

Felsenstein J. (1989) PHYLIP Phylogeny Inference Package, Cladistics 5, 164–166.

Fristensky, B. (1999) Building a multiuser sequence analysis facility using freeware, in: Bioinformatics Methods and Protocols, (Misener, S. and Krawetz, S., eds.), Humana Press, Totowa, NJ, pp. 131–145.

Smith, S., Overbeek, R., Woese, C. R., Gilbert, W., and Gillevet, P. M. (1994) The Genetic Data Environment: an expandable GUI for multiple sequence analysis, Comp. Appl. Biosci. 10, 671–675.

(*see* Website: http://megasun.bch.umontreal.ca/pub/gde/)

Staden R., Beal, K. F., and Bonfield, J. K. (1999) The Staden Package, in: Bioinformatics Methods and Protocols , (Misener, S. and Krawetz, S, eds.), Humana Press, Totowa, NJ, pp. 115–130.