Xpression
Harwood Lab
June 27 2011
Colin Lappala

Introduction

General Assumptions

The purpose of this guide is to create a Unix environment for Xpression by installing the software it requires to run. Xpression is available for systems that its dependencies are available for, which means only Unix-like operating systems such as Linux and MacOSX. All the required software is free, readily available, and simple to install. Some of the software may already be installed on your system, but for the sake of completeness no assumptions

are made about what is already installed, except where noted. The guide will use system utilities common to Unix/MacOSX systems to install all software from source. No previous command-line knowledge is necessary, but a basic computer competency and proper caution when performing system-wide super-user tasks is required.

Formatting

Throughout this guide, instruction will be interspersed with commands and arguments that need to be entered into the terminal. This guide also assumes BASH (Bourne Again SHell) is the default and a terminal emulator is already installed. Common terminals are gnome-terminal, xterm, Konsole and rxvt in Linux and Terminal in MacOSX. BASH displays a prompt that in this guide is represented as a '$' symbol.

Commands are listed as they should be typed in, one per line, with the user pressing the <enter> or <return> key after each command and waiting until the command has finished running before entering the next.

The blank BASH prompt, waiting for input,

```
$
```

The user calls the program `echo` with `hello world` as an argument,

```
$ echo hello world
hello world
```

The program `echo` displays the argument `hello world` and returns,

```
hello world
```

The blank prompt once again, ready to go,

```
$
```

Also, if you need to be in a sepecific directory, or if the previous command changed the directory, the directory will be displayed as the default BASH prompt appears,

```
$ cd ~/some/specific/folder
some/specific/folder$
```

otherwise, the commands can occur elsewhere.

Python has an interactive mode which displays a prompt,

```
>>>
```

which will be shown when the Python being used at the terminal. Since a number of packages for Python as well as Python itself need to be installed, it will be useful later to enter this interactive mode to check for the current installation state before or after installation.

Super user access

It is assumed that the user has "root" or "super-user" access which gives this normal user permission to write to system software folders. If you do not have the sufficient permission, this means you must install the software to directories within your scope of permission, which is likely your home binary folder. If this is the case,or if you want to organise the installation in a non-default way, please see the Appendix for tips on how to accomplish this.

Alternative installation

Installing software from source code may seem intimidating or complicated at first, but with patience and attention to detail, it can be surprisingly useful and even easier than you may have thought previously. It provides you with the most current versions when you want

them, not when packages are created for your system. After performing a few installations, you will notice a pattern emerge: download, unpack, configure, build, install. Each step is relatively simple, but using a package manager can greatly simplify and/or speed up this installation process, in some cases allowing a few clicks with the mouse to handle all the details for you.

Most or all of the software is available as a pre-configured package for many popular Linux distributions and MacOSX. At any step of the installation, the instructions can be circumvented by using a package management system such as Synaptic/Advanced Packaging Tool/dpkg, YUM/RPM, fink, MacPorts, and so on. Providing instruction for these various software exceeds the reasonable scope of this guide; for simplicity all instruction will use standard system tools and source code.

Windows users

Theoretically, Xpression might be able to run on the Linux emulator layer for Windows, Cygwin. This has not been designed, developed, nor tested for, but is up to the user to attempt. However, you could do yourself a favor and just install Linux.

It is getting easier every day to install a modern Linux distribution. Many distributions, like the eminently popular Ubuntu, are fast, stable, secure and have an intuitive, Windows/MacOSX-like user interface. They have a friendly community with free assistance, excellent documentation, and generally avoid much of the pedantry Linux has been known for in the past. A multitude of fast, efficient, and free tools are available for the modern biologist or bioinformatician. Ubuntu can be installed via a variety of methods, including a LiveCD and from within Windows itself. You can find Ubuntu at http://www.ubuntu.com/download.

Required Software

Python 2.x, version 2.4.4+

Python was the natural choice for a tool like Xpression since it strives to be open-source, easy to implement, modify, and reuse, is modular and also reasonably fast. Python is an interpreted, open-source programming language that has powerful C libraries and is system-independent. More information can be found at www.python.com.

numpy 1.0+

A Python package of advanced data structures required by other Python packages.

Biopython 1.51+

A suite of useful bioinformatics modules for dealing with sequence data and common analysis work-flows.

pysam 0.3+

Python wrapper for samtools.

bwa 0.5.7+

Burrow-wheeler aligner, a fast and efficient short read mapper that uses the burrows-wheeler compression algorithm and has a very small memory footprint. It can...

samtools 0.1.10+

Suite of tools to manipulate, filter, and view Sequencing Alignment/Mapping files and their binary equivalent, BAM.

Java Runtime Environment (JRE) 1.5/5.0+

Java is a system-independent, object-oriented programming language that allows cross-platform applications to be created. The GUI uses the Swing framework to provide a consistent experience between platforms. Java is only needed if the GUI will be used.

Check for preinstalled software

This guide assumes that none of the software is previously installed, but you can save time by performing a quick check before installation.

Automatic check

The script ‘`xpression_software_check.sh`’ attempts to automate this process of checking for which required software is already on your system.

Find the place where ‘software_check.sh' is installed and type,

```
$ sh path/to/software_check_folder/xpression_software_check.sh
```

This will result an output of ‘`Pass`’ if the software appears to be installed or ‘`Fail`’ if it does not.

Using the argument/switch '`-v`' as in,

```
$ sh path/to/software_check_folder/software_check.sh -v
```

will provide a list, and '`-h`' gives a brief help section.

Note: If the graphical user interface is not to be used, Java is not needed and its test can be ignored.

Manual check

You can also check for the presence of each piece of software manually at any time. The following example can be used for any of the software, replacing ‘python’ with the name of the program. If the software is a python package, you could enter the python interpreter first, then try ‘`import package-name`’.

To find out if python is already installed, you can check by typing,

```
$ python
```

If you see something displayed that looks like,

```
Python 2.6.5 (r265:79063, May  9 2011, 09:03:54)
[GCC 4.5.0] on linux2
Type “help”, “copyright”, “credits” or “license” for more information.
>>>
```

and exit by pressing `<ctrl>+c` or type,

```
>>> exit()
```

you know that Python is installed and this particular step can be skipped. Make note of the version information, as in ‘Python 2.x.x’.

If you see something like,

```
bash: python: command not found
```

or if the version number is lower than required by Xpression, such as Python version 2.4.3, the software will have to be installed or updated accordingly.

Install required software

During the installation process, the correct output from some of the commands such as make and python setup.py build will result in incomprehensible blurs of gobbledygook streaming past the terminal window. This is normal, please do not adjust your monitor. Anything that stops with the last few lines containing statements like error, traceback, permission denied, or others is the outcome of a problem the program has come up against, but this information can be useful if it does occur. Otherwise, you can assume that proceeding to the next step is OK.

If the packages are available to your system, you can skip this method of installation by using a package manager. Please be sure to still check the version numbers if this route is taken.

Python 2.4.4+

Download source code

This guide will use Python version 2.6.5, a stable secure version available as source. You may use any 2.x series 2.4.4 and up. Make your selection and download Python source code found at http://www.python.org/download/releases/.

Install Python

Unpack the compressed tarball,

```
$ tar xzvf Python-2.6.5.tgz
```

or

```
$ tar xjvf Python-2.6.5.tar.bz2
```

then enter the new directory,

```
$ cd Python-2.6.5
```

automatically configure the installation,

```
Python-2.6.5$ ./configure
```

build installation files,

```
Python-2.6.5$ make
```

finally, install Python,

```
Python-2.6.5$ sudo make install
```

numpy 1.0+

Download source code

Download the source code or the binary/package for your system from either http://new.scipy.org/download.html or more directly http://sourceforge.net/projects/numpy/files/.
This guide will assume installation from the latest source code.
Any issues that may arise may be solved by visiting http://docs.scipy.org/doc/numpy/user/install.html.

Install numpy

After the file has downloaded, unpack the compressed tarball,

```
$ tar xzvf numpy-1.6.0.tar.gz
```

enter the new directory,
```
$ cd numpy-1.6.0
```

Note: For simple instructions or updates refer to the `README` *or* `INSTALL` *files likely present in this directory*
```
numpy-1.6.0$ ls
numpy-1.6.0$ cat README.txt
```

build the package,
```
numpy-1.6.0$ python setup.py build
```

and now install the numpy package with super-user access,
```
numpy-1.6.0$ sudo python setup.py install
```

Enter the python interpreter to check that the install was successful,
```
$ python
Python 2.6.5 (r265:79063, May  9 2011, 09:03:54)
[GCC 4.5.0] on linux2
Type “help”, “copyright”, “credits” or “license” for more information.
```

then import the numpy module,
```
>>> import numpy
```

and check the module version number with,
```
>>> numpy.__version__
‘1.6.0‘
```

exiting by pressing `<ctrl>+c` or type,
```
>>> exit()
```

Biopython 1.51+

An excellent guide for installing this versatile, powerful bioinformatics package can be found at http://biopython.org/DIST/docs/install/Installation.html.

Download Source Code

Download the latest source code from http://www.biopython.org/wiki/Download making sure the version number is 1.51 or higher. This wiki is also another useful resource for installation instructions.

After the package is downloaded, unpack the tarball,
```
$ tar xzvf biopython-1.57.tar.gz
```

and enter the new directory,
```
$ cd biopython-1.57
```

build the package,
```
biopython-1.57$ python setup.py build
```

and now install Biopython with super-user access granted for this command,
```
biopython-1.57$ sudo python setup.py install
```

and finally, check that the installation was successful. Type,
```
$ python
Python 2.6.5 (r265:79063, May  9 2011, 09:03:54)
[GCC 4.5.0] on linux2
Type “help”, “copyright”, “credits” or “license” for more information.
>>>
```

then import the Bio module with,
```
>>>import Bio
```

and check the version,
```
>>> Bio.__version__
‘1.57‘
```

pysam 0.3+

Download source code
Download the latest pysam source code from http://code.google.com/p/pysam/downloads/.

Install pysam
After the download completes, unpack the tarball,
```
$ tar xzvf pysam-0.4.1.tar.gz
```

enter the new directory,
```
$ cd pysam-0.4.1
```

build the package,
```
pysam-0.4.1$ python setup.py build
```

and finally install pysam,
```
pysam-0.4.1$ sudo python setup.py install
```

Check for a successful install by typing,
```
$ python
Python 2.6.5 (r265:79063, May  9 2011, 09:03:54)
[GCC 4.5.0] on linux2
Type “help”, “copyright”, “credits” or “license” for more information.
>>> import pysam
>>> pysam.__version__
‘0.4.1‘
```

bwa 0.5.7+

Download source code
Download the latest source code from http://sourceforge.net/projects/bio-bwa/files/.

Install bwa
Uncompress the tarball,
```
$ tar xjvf bwa-0.5.9.tar.bz2
```

enter the new directory,
```
$ cd bwa-0.5.9
```

build the binary files,
```
bwa-0.5.9$ make
```

*c*opy the executable `bwa` and any perl scripts to a directory in your `$PATH` variable,
```
bwa-0.5.9$ cp ./*.pl ./bwa /home/usrname/bin/
```
where `usrname` is your username,
or,
```
bwa-0.5.9$ sudo cp ./*.pl ./bwa /usr/local/bin
```

Note: This installation directory can be anything listed in the system `$PATH` *variable. For additional help, please see the appendix.*

check the install,
```
$ bwa
Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.5.9-r16
Contact: Heng Li <lh3@sanger.ac.uk>
...
```

samtools 0.1.10+

Download source
Download the latest source code from http://sourceforge.net/projects/samtools/files/

Install samtools
Uncompress the tarball,
```
$ tar xjvf samtools-0.1.16.tar.bz2
```

enter the new directory,
```
$ cd samtools-0.1.16
```

Note: See `INSTALL` *in this directory for more installation details*

build executable files,
```
samtools-0.1.16$ make
```

copy `samtools` and all executables in the directory '`misc`' to a directory in your `$PATH`,
```
samtools-0.1.16$ cp ./samtools misc/*.pl misc/*.py /home/usrname/bin/
```
where `usrname` is your username.
or,
```
samtools-0.1.16$ sudo cp ./samtools misc/*.pl misc/*.py /usr/local/bin
```

Note: This installation directory can be anything listed in the system `$PATH` *variable. For additional help, please see the appendix.*

check install
```
$ samtools
Program: samtools (Tools for alignments in the SAM format)
Version: 0.1.12a (r862)
...
```

Java JRE 1.5/5.0+ for the Graphical User Interface

Download a file from http://www.java.com/en/download/ and follow the instructions found on this page. A self-extracting binary is likely the easiest method next to using your system's package management system if a package is available.

For instance, the latest self-extracting binary for the 64-bit architecture at the time of writing this guide was `jre-6u26-linux-x64.bin`. After downloading, the instructions found at http://www.java.com/en/download/help/linux_x64_install.xml were followed to install java to a directory, and the $PATH variable altered to include the binary java found in the bin

folder in the jre install folder. Alternatively a symoblic link could be created from a folder in the system path to this java binary.

As an overview and supplement to the instructions found on Java's website,

change the directory to the path to which you want to install. This could be /usr/local/bin or a user-specific folder such as,
```
$ cd ~/java
```

make the downloaded file executable with chmod,
```
~/java$ chmod +x ~/downloads/jre-6u26-linux-x64.bin
```

execute the binary file,
```
~/java$ ~/downloads/jre-6u26-linux-x64.bin
```

make sure you are in a directory that is listed in the $PATH variable and create a symbolic link to the installation folder from that directory (ie, /usr/local/bin to /home/usrname/jre-<version>/bin/java),
```
~/java$ cd /usr/local/bin
/usr/local/bin$ ln -s ~/java/jre1.6.0_26/bin/java java
```

After installation, check java install,
```
$ java -version
java version "1.6.0_25"
Java(TM) SE Runtime Environment (build 1.6.0_25-b06)
Java HotSpot(TM) Server VM (build 20.0-b11, mixed mode)
```

Appendix

A quick and painless guide to the command-line

This is provided as an overview for those who have never used the command-line before. Using the command-line can be daunting and frustrating at first, but it has trade-offs such as increasing the amount of output from a program, including errors, as well as greater efficiency at many mundane file operations and tasks. For many uses, the command-line actually makes doing work much easier and more efficient in the long run.

This extremely brief introduction will include everything you need to know for the purposes of the installation and a few tips to make it easier on you. It should apply to either Linux or MacOSX, since the same basic tools exist on both systems.

The terminal / terminal-emulator

A terminal or terminal emulator is a program that uses text-based commands to interact with and display information about the computer system. The guide assumes one is already installed and that it is open and ready for input at each step. Terminal entries are listed in the guide on as a prompt symbol followed by a command and possibly example output from running the command. Each command is on a separate line, implying the 'enter' or 'return' button is pressed between each one, and the result of the command is completed before typing the next one.

Although shell scripting is kept to a minimum with Xpression, the installation requires basic interaction with the system shell and BASH (Bourne Again SHell) is expected as the default. This is likely to be the default for your system, but if you want to check you can type:
```
$ echo $SHELL
```

which in this case produces,
`/bin/bash`

Basic file structure shorthand
`~` This is the symbol for the user's home folder, which is modifiable by the user without restriction.
`/` This symbol represents the system's root folder, which contains all the system files and programs which (except for the home folder) are not modifiable by the user, who must get permission to do so.
`.` This stands for the current directory.
`..` This means the directory above the current, 'up' a directory.

`<tab>` **to auto-complete an entry**
This makes using a terminal bearable. If two files exist in the current directory with the names '`gi_one_emb_number.gbwithparts`' and '`gi_another_number_emb_number.gbwithparts`', typing,
`$ gi_`
then pressing the `<tab>` button will fill in this name if it is unique beyond that point.

Since it is not unique, pressing `<tab>` again (two times total) will display a list of the possibilities,
```
$ gi_<tab><tab>
gi_one_emb_number.gbwithparts
gi_another_number_emb_number.gbwithparts
```

Then entering '`o`' will make the entry unique to the filenames in the local directory and by pressing `<tab>`, will fill in the remainder,
```
$ gi_o<tab>
$ gi_one_emb_number.gbwithparts
```

This can apply to many things on the system, including directories, files and programs.

`cd` **to change directory**
cd is a system tool to change the current working directory.
type,
```
$ cd some/navigable/path
some/navigable/path $
```
Now the current directory is `some/navigable/path`

Using tab-completion is very useful here, as typing every last letter to that file far, far away is not necessary.

Using the shortcuts to system folder can make navigation much faster also.

To move up a directory, type,
`$ cd ..`

while typing,
`$ cd ~`
brings you to your home directory,

and typing,
`$ cd /`
brings you to the root directory

to return to the most previous directory, you can type,
`$ cd -`

`ls` to list directory information
ls is a program that displays the current directory or the directory specified.

To display the contents of the current directory, type,
`$ ls`
(which is the same as typing, '`ls .`')

and to display the contents of another directory without moving anywhere, typing,
`$ ls ./path/to/somewhere`
will list the contents of the folder `./path/to/somewhere`.

Installation Paths
All installed software must be visible to the system no matter what current directory you are in. When you enter a command, the system looks at the contents of the `$PATH` variable which lists directories that contain executable files, such as `cd` and `echo` in the `/bin` directory.

To view the directories currently in the system path, type,
`$ echo $PATH`

For example, the `$PATH` variable contains these directories on the system this guide is written from,
`/home/colin/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/bin/core_perl`

Each directory is separated by a colon, '`:`'. The directories, `/bin` and `/usr/bin`, apply to all users of the system. Since altering the contents of these paths could result in serious damage to the system, a single normal user cannot edit them, at least not without special permission. `sudo` is a commonly installed program that temporarily grants this permission, or root-access, to the user and simply needs to precede any critical command.

```
$ make-sandwich
[Only root can do that]
$ sudo make-sandwich
Password:
[OK, root access granted]
```

So for the installation or copying of a program to a system-wide location, you simply need to precede that command with `sudo` and enter your user password to allow this command to have temporary permission to modify system-wide folders or settings.

Modifying the `$PATH` variable
If you do not have super-user permission or if you wish to organise the installation differently, you can edit `.bash_profile` or `.bashrc` located in the current user's home (`~`) directory. Changes from editing `.bash_profile` will take effect when the user logs back in, while `.bashrc` is checked every time a terminal is opened.

To edit the `~/.bash_profile` or `~/.bashrc` file, open it with the text editor `nano`,
`$ nano ~/.bash_profile`
or
`$ nano ~/.bash_rc`

then add the lines you need to the file, for example,
```
PATH=$PATH:/home/usrname/bin:/home/usrname/scripts
export PATH
```

where `usrname` is your username, and making sure to separate each path with a colon. Now save the file, which in `nano` is acomplished by pressing `<ctrl>` and `o` together.

If `.bash_profile` was modified, you will have to log out, and then log back in to see these changes take place.

If you do have super user access, and want to have these changes apply for all users except root, the `/etc/profile` file may be altered instead.

Lets check the $PATH variable again to make sure the changes took place,

```
$ echo $PATH
/bin:/usr/bin:/home/colin/bin
```

Now a program installed into any of these listed directories can be accessed from anywhere on the system.