

# AlloSystem Install Tutorial

Lance Putnam and Lars Knudsen

July 19, 2013

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Supported Operating Systems . . . . .	2
<b>2</b>	<b>Get Development Tools</b>	<b>2</b>
2.1	Linux Ubuntu . . . . .	2
2.2	Mac OS X . . . . .	2
2.3	Windows and MinGW/MSYS . . . . .	5
<b>3</b>	<b>Install AlloSystem</b>	<b>7</b>
<b>4</b>	<b>Build-and-Run Applications</b>	<b>8</b>
<b>5</b>	<b>GUI and Sound</b>	<b>8</b>
<b>6</b>	<b>Advanced Configuration</b>	<b>9</b>
6.1	Custom Project Directories . . . . .	9
6.2	Linking to Other Libraries . . . . .	9
6.3	Compiling Your Own Object Code . . . . .	10
<b>7</b>	<b>Troubleshooting</b>	<b>11</b>
7.1	CasE Sensitivity . . . . .	11
7.2	I Can't Find The '~' Directory! . . . . .	11
7.3	'No such file or directory' . . . . .	11
7.4	make: Nothing to be done for 'xxx'. . . . .	11
7.5	make: *** No rule to make target 'xxx'. Stop. . . . .	11
7.6	MinGW/MSYS: Spaces in User Name . . . . .	11
7.7	MinGW/MSYS: Permission Denied . . . . .	11
7.8	Bash Script (.sh file) Does Not Execute (Properly) . . . . .	12
7.9	'<cmath> not found' When Compiling Sources . . . . .	12
7.10	Problems With Multiple Versions of Xcode . . . . .	12

# 1 Introduction

This tutorial explains how to install AlloSystem under Linux Ubuntu, Mac OS X, and Windows. The main means of operating AlloSystem is through a Unix-like shell and GNU tools GCC and make. The first part of the tutorial explains how to set up a Unix environment and install the necessary tools. The second part describes how to install AlloSystem. The last part explains how to compile and run custom applications using AlloSystem.

## 1.1 Supported Operating Systems

Currently, the following operating systems are known to work with AlloSystem:

- Linux Ubuntu
- Mac OS X 10.5.x (Leopard), 10.6.x (Snow Leopard), 10.7.x (Lion), 10.8.x (Mountain Lion)
- Windows 7

For Windows users, it is highly recommended to install Linux Ubuntu. This can be done easily and safely using Wubi. If you choose to use operating systems or tools other than those recommended in this tutorial, then *you are on your own*.

## 2 Get Development Tools

The first step is to obtain the development tools necessary to compile and run AlloSystem programs. Skip to the platform you are using and follow the instructions there.

### 2.1 Linux Ubuntu

Ensure that both make and g++ are installed by entering the following command in the terminal:

---

```
$ sudo apt-get install make g++
```

---

When that finishes, proceed to 3.

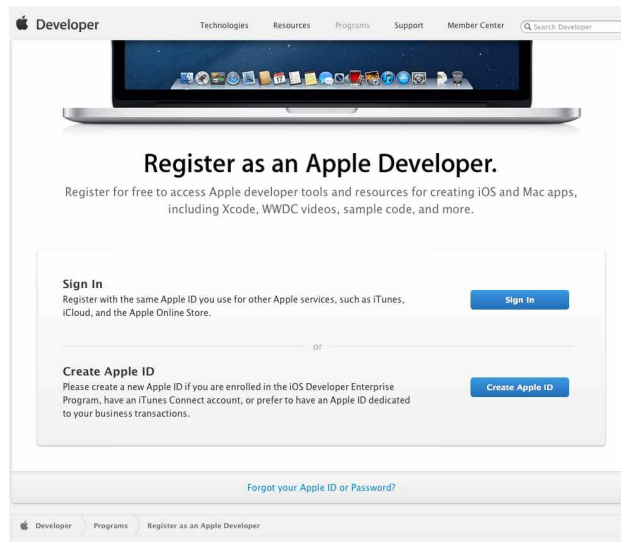
### 2.2 Mac OS X

#### Before Beginning

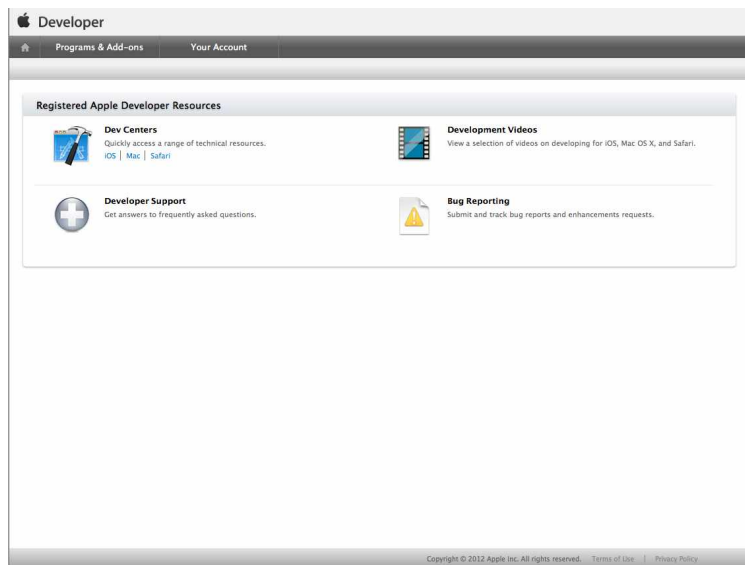
Before beginning to install the development tools, it is crucial that you complete any OS updates. If you don't, the installation of Xcode may fail with an "unknown error". (This is a known issue with OS X 10.6.x Snow Leopard and Xcode 3.2.6.) Do system updates by clicking the Apple symbol in the upper-left corner of the screen and then selecting "Software Update...". Let the updater finish its course which may require restarting. Repeat this update process until there are no more updates.

#### Get Apple Development Tools

To get the Apple development tools, start by going to <http://developer.apple.com/programs/register>. Click the 'Sign In' button to login or create a new account.




Under 'Dev Centers' click 'Mac'.



Scroll down and click 'View all downloads'.

### Downloads



**Xcode 4.4.1**  
This is the complete Xcode developer toolset for Mac, iPhone, and iPad. It includes the Xcode IDE, iOS Simulator, and all required tools and frameworks for building OS X and iOS apps.

Posted Date: August 7, 2012  
Build: 4F1003  
Included iOS SDK: iOS 5.1  
Included Mac SDK: Mac OS X 10.8

[View in Mac App Store](#)


#### Additional Downloads

Find additional pre-release software, previous versions of Mac OS X, and hardware driver downloads in the Apple Developer Downloads site.

[View all downloads](#)


### What's New in OS X Mountain Lion

See what's new in OS X and build the most innovative Mac apps ever.




### WWDC 2012 Videos

Watch over 100 sessions covering the latest innovations in iOS and OS X.




### Software Licensing & Trademarks

Use Apple Software, technologies and trademarks in your product.



### iCloud for Developers

Learn how to integrate iCloud Storage APIs into your apps.



Developer

Mac Dev Center

Technologies

Developer Tools

iOS

OS X

Safari

Resources

iOS Dev Center

Mac Dev Center

Safari Dev Center

App Store Resource Center

Developer Forums

Development Videos

iAd

Apple Applications

Hardware & Drivers

Licensing & Trademarks

iPod, iPhone & iPad Cases

Open Source

Programs

iOS Developer Program

iOS Enterprise Program

iOS University Program

Mac Developer Program

Safari Developer Program

MFi Program

Support

iOS Developer Program

Mac Developer Program

Safari Developer Program

Bug Reporting

iTunes Connect Support

Technical Support


This takes you to a search page.

1 - 20 of 141

Page 1 of 8

#### Categories

- ☒ Applications (10)
- ☒ Developer Tools (122)
- ☒ iOS (4)
- ☒ OS X (49)
- ☒ OS X Server (9)

Description	Release Date
<b>Hardware IO Tools for Xcode - Late July 2012</b> This package includes tools for developing for specific hardware and external devices. These tools were formerly bundled in the Xcode installer, and include: PacketLogger, ICRRegistryExplorer, Bluetooth Explorer, and the Network Link Conditioner pane for System Preferences. Tools in this package support running on OS X Mountain Lion. <div>  Hardware IO Tools for Xcode 4.4 - Late July 2012.dmg (28.86 MB) </div>	Aug 22, 2012
▶ <b>Command Line Tools (OS X Lion) for Xcode - August 2012</b>	Aug 7, 2012
▶ <b>Command Line Tools (OS X Mountain Lion) for Xcode - August 2012</b>	Aug 7, 2012
▶ <b>Xcode 4.4.1</b>	Aug 7, 2012
▶ <b>Graphics Tools for Xcode - Late July 2012</b>	Aug 7, 2012
▶ <b>Dashcode for Xcode - Late July 2012</b>	Aug 7, 2012
▶ <b>Audio Tools for Xcode - Late July 2012</b>	Aug 7, 2012
▶ <b>Auxiliary Tools for Xcode - Late July 2012</b>	Aug 7, 2012
▶ <b>Glossaries</b>	Aug 6, 2012
▶ <b>AppleGlot 3.4</b>	Aug 6, 2012
▶ <b>IIOUSBFamily Log Release for OS X 10.8</b>	Aug 1, 2012
▶ <b>Command Line Tools (OS X Lion) for Xcode - July 2012</b>	Jul 27, 2012
▶ <b>Command Line Tools (OS X Mountain Lion) for Xcode - July 2012</b>	Jul 27, 2012
▶ <b>CrashWrangler</b>	Jun 22, 2012
▶ <b>Java for OS X 2012-004 Developer Package</b>	Jun 19, 2012

Enter 'Xcode' into the search bar and get the latest version of Xcode *and* Command Line Tools that work for your version of OS X. (Note that with older versions of Xcode the command line tools are included with Xcode.) Unfortunately, every major version of OS X seems to have a new and backwards-incompatible version of Xcode. Use the following table to determine which version of Xcode and Command Line Tools to install with what version of OS X:

OS X Version	What To Install
10.5 (Leopard)	Xcode 3.1.4 Developer Tools
10.6 (Snow Leopard)	Xcode 3.2.6 and iOS SDK 4.3
10.7 (Lion)	Xcode 4.4.1, Command Line Tools (OS X Lion) - August 2012
10.8 (Mountain Lion)	Xcode 4.4.1, Command Line Tools (OS X Mountain Lion) - August 2012

Table 1: What tools to install for which version of OS X

After downloading the disk images (.dmg file), double-click them to mount them (if not done automatically). If you downloaded either Xcode 3.1.4 or Xcode 3.2.6, double-click the appropriate package (.pkg) file inside the mounted drive to run the installer. If you downloaded Xcode 4.4.1, then drag-n-drop the Xcode icon into the /Applications folder and then run the Command Line Tools installer. When installation is complete, you will have both an editor and all the command-line tools we will be using such as `make` and `g++`.

### Install Package Manager

We will use Macports, a package management tool for OS X, to install AlloSystem dependencies. Download and install the latest version from <http://www.macports.org/install.php>. Make sure to install the correct version for your version of OSX (Leopard, Snow Leopard, Lion, or Mountain Lion).

## 2.3 Windows and MinGW/MSYS

### Install Code Editor

You will need to install an editor that recognizes and syntax colorizes source code. We recommend either Visual Studio Express or Notepad++. Visual Studio is a full-blown IDE while Notepad++ is a lightweight text editor. Notepad++ is preferred as it starts up much more quickly.

### Install MinGW and MSYS

MinGW is an emulation of the GCC compiler suite. MSYS provides a Unix-like CLI and commands (`bash`, `make`, `grep`, etc.). Download and run the latest version of the MinGW installer `mingw-get-inst-xxxxxxx.exe` from <http://sourceforge.net/projects/mingw/files/Installer/mingw-get-inst/>. Be careful when installing not to install MinGW in a path containing spaces, such as `C:\Program Files (x86)\MinGW`, as this will cause a lot of problems later on. The rest of the tutorial assumes `C:\MinGW` is the install directory. If you choose another install directory just replace any following instances of `C:\MinGW` with your path. When asked to "Select Components", choose

- C Compiler
- C++ Compiler
- MSYS Basic System

Finish the MinGW installer.

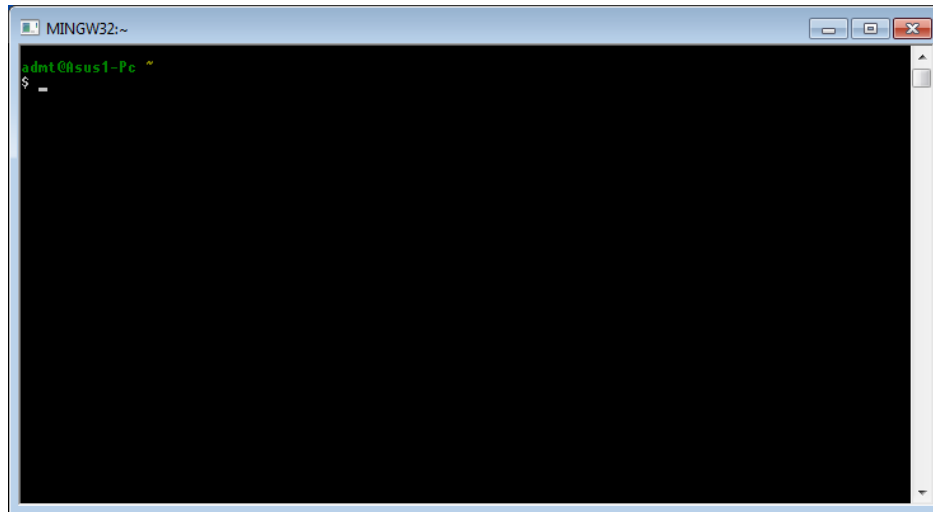
Next, bring up a DOS prompt. Do this by clicking the Windows button in the lower-left corner, typing `cmd.exe` into the search bar, and pressing enter. At the DOS prompt, execute the following commands:

```
cd C:\MinGW\bin
mingw-get update
mingw-get install msys-wget msys-unzip msys-man
```

This will install additional programs for MSYS that we will need later to install dependencies. When the install command is finished, close the DOS prompt window and move on to the next step.

## Start MSYS

Open a new MSYS shell (not to be confused with a DOS shell!) by launching MSYS from the Start menu or desktop or by running `C:\MinGW\msys\1.0\msys.bat`. The MSYS shell looks like this:



By default, the MSYS directory `C:\MinGW\msys\1.0` will be mounted as root directory `/` and as `/usr/`. The first time you run MSYS it will create a home directory in `C:\MinGW\msys\1.0`. The `~` (user home) directory will be something like `C:\MinGW\msys\1.0\home\<username>`. Traditional Windows drives, like `C:` and `D:`, can be accessed as `/c` or `/d`, respectively. Use the `mount` command to get an overview of all the drive mappings.

### 3 Install AlloSystem

At this point, you should have the appropriate development tools installed. In the following instructions, when we say 'Unix shell' we mean MSYS for Windows users, and the 'Terminal' application for Mac OS X and Linux users.

First, create a new directory `~/code/`. This can be done from the Unix shell with the command

```
$ mkdir ~/code
```

Next, download the AlloSystem source code <https://github.com/AlloSphere-Research-Group/AlloSystem/archive/devel.zip>. Within the .zip file should be another directory having a name of something like `AlloSystem-xxx`. You should move this directory to the directory `~/code/` you just made. After the directory `AlloSystem-xxx` is moved, rename it to `allosystem`.

From the Unix shell, 'cd' into the `~/code/allosystem/` directory. If you do 'ls' you should see something like:

```
$ ls
Makefile      alloGLV      allovsvr
Makefile.buildandrun  allocore     doc
Makefile.common  allocv       linux
Makefile.rules   allonect     osx
README.md       alloutil     work
```

Next, you need to install the necessary dependencies for AlloSystem. (Windows users only: you must *deactivate all virus checkers and antispyware programs, such as Windows Defender*. These monitoring programs are known to terminate or halt important configuration processes that take place in the following steps. If you do not disable them, then the libraries will not get installed properly.) Install dependencies with the following commands (notice if you are using MinGW/MSYS then you don't need the `sudo` in front of the command):

```
$ cd allocore
$ sudo ./install_dependencies.sh
$ cd ..
$ cd alloutil
$ sudo ./install_dependencies.sh
$ cd ..
```

If prompted for your password, enter the same password you use to log in to the OS. Installing all the dependencies may take some time, so please be patient. After dependencies are installed, the next step is to build the AlloCore and AlloUtil modules with the following commands:

```
$ make allocore
$ make alloutil
```

To check if the system is installed successfully you can build an example by navigating to the `~/code/allosystem/allocore` folder and then running the following command:

```
$ make examples/av/simpleApp.cpp
```

If a ball with oscillating color shows up, then your system is configured correctly.

## 4 Build-and-Run Applications

While AlloSystem is intended to be used mainly as a library, it does provide a simple means of building and running custom applications that link to AlloSystem. It is flexible enough to be used all the way from building simple prototypes to creating complex applications. This section describes how to setup and use AlloSystem's "build-and-run" capability.

By default, there are two directories recognized by the AlloSystem build-and-run facility: `examples/` and `work/`. The directory `examples/` contains example source code for libraries. The directory `work/` is for user application source files and thus will be of primary concern here. You can place any source (`.cpp`) files having a `main()` function in the `work/` directory (or any subdirectories thereof) and compile and run the source with the command

---

```
$ make work/myfile.cpp
```

---

Remember, you must be in `~/code/allosystem/` when running the `make` command. The executable binary is placed in `build/bin/` and has the same name as the source file. If you would only like to build the application, without also running it, then you can append `AUTORUN=0` to the line above, i.e.,

---

```
$ make work/myfile.cpp AUTORUN=0
```

---

## 5 GUI and Sound

The following instructions are only required if you would like to use graphical user interface (GUI) or sound synthesis components in conjunction with AlloSystem. We use GLV as the GUI toolkit and Gamma as the sound synthesis library. Download the source code of each at <https://github.com/AlloSphere-Research-Group/GLV/archive/master.zip> and <https://github.com/LancePutnam/Gamma/archive/devel.zip>. Unzip them within the directory `~/code` (alongside where you installed AlloSystem). Rename the GLV and Gamma directories to GLV and Gamma. From the shell, this will look like

---

```
$ cd ~/code/  
$ ls  
allosystem  
Gamma  
GLV
```

---

Next, build GLV and Gamma using the following commands:

---

```
$ cd ~/code/allosystem  
$ make Gamma  
$ make GLV  
$ make alloGLV
```

---



## 6 Advanced Configuration

### 6.1 Custom Project Directories

It is possible to configure AlloSystem to use your own custom project directory for its build-and-run facility. We will call this your *user directory* and refer to it in this document generically as `<userdir>` from this point onward. The actual name of the user directory can be anything you choose as long as it is a valid path, e.g., `myprojects`, `mycode` or your initials, such as `ljp`. Start by creating the directory `~/code/allosystem/<userdir>/`. Next, make a copy of the file `Makefile.usertemplate` and name it `Makefile.user`. `Makefile.user` must be in the folder `~/code/allosystem/` along with `Makefile.usertemplate`. You can do this from the terminal with the commands

---

```
$ cd ~/code/allosystem/  
$ cp Makefile.usertemplate Makefile.user
```

---

Now, open `Makefile.user` in an editor and change the line reading

---

```
#RUN_DIRS += foo/
```

---

to

---

```
RUN_DIRS += <userdir>/
```

---

using the same directory name you created in the previous step. Notice the `#` (comment symbol) is removed. Also, do not forget the trailing slash after `<userdir>`. Relative paths in `RUN_DIRS` are relative to the `allosystem/` directory. At this point, AlloSystem will recognize your user directory as a valid build-and-run directory. You can add as many user directories as you like to `RUN_DIRS` remembering to separate them with spaces.

While it is possible to have as many user build-and-run directories, sometimes it is convenient to be able to build and run sources from other directories without always having to add them to the `RUN_DIRS` variable. This can be accomplished by creating a symbolic link in one of your valid build-and-run directories:

---

```
$ cd ~/code/allosystem/  
$ ln -s sourcedir/ <userdir>/targetdir
```

---

This creates a symbolic link to `sourcedir/` that is treated like the directory `<userdir>/targetdir/`.

### 6.2 Linking to Other Libraries

If your projects need to link to libraries that are not already used by AlloSystem, then there is some additional configuration required. You can configure this in one of two ways—globally, where all projects link to specified libraries, or locally, where only a specific project links to specified libraries. To set up global linking, simply add the linker flags to your existing `Makefile.user` file. For example,

---

```
# Somewhere in Makefile.user ...  
  
# Append directories to be searched for include files  
CPPFLAGS += -I/usr/include/  
  
# Append directories to be searched for library files  
LDFLAGS += -L/usr/lib/
```

---

```
# Append libraries to be linked to
LDFLAGS += -lfftw3f -lMagick++
```

---

If you would only like local linking, that is, linking to libraries for only one project, then place all the project's source files along with a file `flags.txt` in a subdirectory of one of your user directories. Then, in `flags.txt`, add all the necessary linker flags in the same way as done above with `Makefile.user`.

### 6.3 Compiling Your Own Object Code

There comes a time when you would like many separate projects to reuse your own custom code, akin to using a library. For example, you may have several utility functions or classes that you would like all projects to have access to. Let's say you have the two files `myutils.h` and `myutils.cpp` that contain your custom functions and classes. At this point, all that is necessary is to place these files in some directory, say `~/code/allosystem/myutils/`, and add the following line to your existing `Makefile.user` file:

```
RUN_SRC_DIRS += myutils/
```

---

## 7 Troubleshooting

### 7.1 Case Sensitivity

If your commands are not working, it is possible that the case of one or more characters in the command is wrong. Many Unix systems are case sensitive.

### 7.2 I Can't Find The '~' Directory!

The character '~' designates the home directory of the current user. This is usually where you start out when you first open a terminal. The exact location of ~ varies by platform. If you want to know exactly where it is, enter the following in the terminal

---

```
$ file ~
```

---

### 7.3 'No such file or directory'

This error message is typically the result of misspelling a file or directory name or being in the wrong directory. Always use tab autocompletion when typing file/directory names to ensure that they exist and are spelled correctly.

### 7.4 make: Nothing to be done for 'xxx'.

This means all dependencies of the rule you specified are up-to-date.

### 7.5 make: \*\*\* No rule to make target 'xxx'. Stop.

The Makefile does not contain the rule you specified. The rule is the thing you are specifying after typing make. If you are attempting to build and run a source file, then this message means that the directory of the source file does not exist where you specified. A common cause is that you are in the wrong directory. Always use tab autocompletion to ensure that you have file paths correct. Another cause is that you are trying to build and run a source file that is not in work/ or one of your build-and-run user directories (6.1).

### 7.6 MinGW/MSYS: Spaces in User Name

MSYS simply does not like user names with spaces in them. If your user name contains spaces, then it will create problems with many of the build tools. To solve the problem, you must change your user name in MSYS. Add the line

---

```
set USERNAME=<myname>
```

---

to the beginning of C:\MinGW\msys\1.0\msys.bat where <myname> does not have spaces.

### 7.7 MinGW/MSYS: Permission Denied

If you see a message 'Permission denied', it is possible that the Windows Application Experience service is not configured properly. Usually, logging out of Windows and logging back in fixes the problem (no need to restart Windows). If the problem persists, you will need to manually configure Windows group policies.

To configure group policies, click the Start button and in the search bar type: gpedit.msc. Click this file to start the Local Group Policy Editor. Next, navigate to Local Computer Policy -> Computer Configuration -> Administrative Templates -> Windows Components -> Application Compatibility and do the following

- Select Turn Off Application Compatibility Engine
- Select Enabled under the Settings tab
- Select Turn off Program Compatibility Assistant
- Select Enabled under the Settings tab

See <http://www.blackviper.com/windows-services/application-experience/> for more information.

## 7.8 Bash Script (.sh file) Does Not Execute (Properly)

If you have problems trying to execute a Bash script (.sh file), then it is possible that its file mode is incorrect. Make the script executable by running

---

```
$ chmod u+x <filename>.sh
```

---

## 7.9 '<cmath> not found' When Compiling Sources

This occurs for users with OSX 10.7 and above who have installed Xcode 4.4.1 and above. The cause is that Xcode is not found by the build system. To fix the problem, ensure that Xcode.app is placed in /Applications and not ~/Applications or some other folder. If you cannot navigate to /Applications from Finder, then open a Terminal and type

---

```
$ open /Applications
```

---

## 7.10 Problems With Multiple Versions of Xcode

If you have older versions of Xcode installed and want to remove them (i.e., to save space or fix installation problems) you can uninstall them by running in a terminal

---

```
$ sudo /Developer/Library/uninstall-devtools --mode=all
```

---

Typically, you will want to remove old versions *before* trying to install a newer version. If you want to keep multiple versions of Xcode installed on your machine, you can switch between them by running

---

```
$ sudo xcode-select -switch [path to Xcode]
```

---

where the path after -switch is the version of Xcode you want to use, e.g. /Applications/Xcode.app/.