- revision history for given file (how many additions/deletions made by developer for file)

- "visit" children of block/statement node to get conditional expressions/null literals (DONE)

-

Get revision history in the repository (filter out each name and make a new developer object)

For each developer, create a list of files in the repository they have modified (will eventually be useful when looking at files with conditionals/null checks)

- fb-contrib download, pull out stuff you don't need but keep others (i.e. build system)

-- To view human friendly version of bytecode:

javap -c MyFile.class > ./out.txt

Later:

-- diff to show what was modified between two consecutive versions of a file (where a null check is present?)

# Notes

## Git Revision History

git log --pretty=format:"%h %cn %cd" --name-only [filename]> output\_project-name.txt

%h = abbreviated hash  
%an = author name  
%cn = committer name

--name-only = list of files modified after the commit information  
--name-status = show list of files affected with added/modified/deleted information as well

-(n) = last n commits   
--author = only commits where author entry matches specified string  
--committer = only commits committer entry matches specified string

## Process Builder

Here's a simple use of ProcessBuilder that duplicates the functions of the DoRuntime example:

|  |
| --- |
| **import**java.io.\*;    **import**java.util.\*;        **public class**DoProcessBuilder {      **public static void**main(String args[]) **throws**IOException {         **if**(args.length <= 0) {          System.err.println("Need command to run");          System.exit(-1);        }         Process process = **new**ProcessBuilder(args).start();        InputStream is = process.getInputStream();        InputStreamReader isr = **new**InputStreamReader(is);        BufferedReader br = **new**BufferedReader(isr);        String line;         System.out.printf("Output of running %s is:",            Arrays.toString(args));         **while**((line = br.readLine()) != **null**) {          System.out.println(line);        }       }     } |

> java DoProcessBuilder ls

Output of running ls is:DoProcessBuilder.class

DoProcessBuilder.java

DoRuntime.class

DoRuntime.java

Notice that the following two lines in DoRuntime:

Runtime runtime = Runtime.getRuntime();

Process process = runtime.exec(command);

were changed to the following line in DoProcessBuilder:

Process process = new ProcessBuilder(command).start();

The ProcessBuilder class has two constructors. One constructor accepts a List for the command and its arguments. The other constructor accepts a variable number of String arguments.

public ProcessBuilder(List command)

public ProcessBuilder(String... command)

With ProcessBuilder, you call start() to execute the command. Prior to calling start(), you can manipulate how the Process will be created. If you want the process to start in a different directory, you don't pass a File in as a command line argument. Instead, you set the process builder's working directory by passing the File to the directory() method:

public ProcessBuilder directory(File directory)

There isn't an obvious setter type method in ProcessBuilder for setting environment variables. Instead, you get a Map of the variables through the environment() method, then you manipulate the Map:

ProcessBuilder processBuilder = new ProcessBuilder(command);

Map env = processBuilder.environment();

// manipulate env

The options for manipulating the environment include adding environment variables with the put() method, and removing them with the remove() method. For example:

ProcessBuilder processBuilder = new ProcessBuilder(

command, arg1, arg2);

Map env = processBuilder.environment();

env.put("var1", "value");

env.remove("var3");

After the environment variables and directory are set, call start():

processBuilder.directory("Dir");

Process p = processBuilder.start();

You can also clear() all the variables from the environment and explicitly set the ones you want.

With methods such as environment() for adding and removing environment variables from the process space, and start() for starting a new process, ProcessBuilder should make it easier to invoke a subprocess with a modified process environment.

You can get the initial set of environment variables by calling the getenv() method of System. Understand that not all platforms support changing environment variables. If you try to change an environment variable on a platform that forbids it, the operation will throw either an UnsupportedOperationException or an IllegalArgumentException. Also, when running with a security manager, you'll need the RuntimePermission for "getenv.\*", otherwise a SecurityException will be thrown.

Remember not to forget the start() call after configuring your instance. And, keep using the Process class to manipulate the streams for the process and to get its exit status.

A word of caution about the examples in this tip. It is possible that the examples will deadlock if the subprocess generates enough output to overflow the system. A more robust solution requires draining the process stdout and stderr in separate threads.

## Adding Environment Variables to path (Mac)

* Open up Terminal.
* Run the following command:

|  |  |
| --- | --- |
| 1 | sudo nano /etc/paths |

* Enter your password, when prompted.
* Go to the bottom of the file, and enter the path you wish to add.
* Hit control-x to quit.
* Enter “Y” to save the modified buffer.

That’s it!  To test it, in *new* terminal window, type:

|  |  |
| --- | --- |
| 1 | echo $PATH |

You should see something similar to this (including the path you’ve added!):

|  |  |
| --- | --- |
| 1  2 | MacMini:~ ryan$ echo $PATH  /usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/usr/local/share/npm/bin |

## Here is a screenshot of what my file looks like: http://architectryan.com/images/2012/10/image.png

## Using JGit (instead of ProcessBuilder)

The most popular way of interacting with JGit involves using the [Git](http://download.eclipse.org/jgit/docs/latest/apidocs/org/eclipse/jgit/api/Git.html) class to wrap a repository and to provide a set of *porcelain* commands. This is a set of commands that roughly mirror the high-level commands that are given at the command line; for example, .add() or .log().

Level2.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | import org.eclipse.jgit.api.Git  Git git = Git.open(new File("/tmp/repo/.git"));  git.clean() ...  git.lsRemote() ...  git.log() ... |

The advantages of using the Git class are that you get to re-use the same repository between invocations, so subsequent commands may be faster. You also have IDE completion and compile time correctness for the arguments, as opposed to the untested strings in the prior examples.

To invoke the command the *builder* pattern is used; the result from the .clean() is actually a CleanCommand. So to invoke it, you need to invoke the .call() method, after providing any necessary arguments:

Level2.java

|  |  |
| --- | --- |
| 1  2 | git.clean().setCleanDirectories(true).setIgnore(true).call();  git.lsRemote().setRemote("origin").setTags(true).setHeads(true).call(); |

Although the builder allows an arbitrary number of arguments to be built up over repeated calls, care must be taken to ensure that any required arguments are set up appropriately.

<http://download.eclipse.org/jgit/docs/latest/apidocs/>  
<http://download.eclipse.org/jgit/docs/latest/apidocs/org/eclipse/jgit/api/LogCommand.html>  
<http://download.eclipse.org/jgit/docs/latest/apidocs/org/eclipse/jgit/revwalk/RevCommit.html>

# Challenges

## Windows 7, Python, virtualenv and the “Unable to find vcvarsall.bat” error [unresolved]

This is because of lacking compile on your system. To solve probem you have two choices: Installing MinGW or installing  [Microsoft Visual Studio Express 2008](http://www.microsoft.com/visualstudio/en-us/products/2008-editions/express)( it is free!).  
  
  
**First Choice:** Install [MinGW](http://sourceforge.net/projects/mingw/files/OldFiles/MinGW-5.0.2.exe/download). After installation add MinGW bin dir to your system path(C:\MinGW\bin). During installation be sure to select "g++". Open a new DOS window and rum command like this:

python setup.py build\_ext -c mingw32

**Second choice:** You'll download "Visual C++ 2008 Express Edition with SP1" and install it. During install it will show an option to install MS SQL Server Express edition, uncheck it, it is not needed for this session. Totally 94 MB will be downloaded from net and you will need about 600MB disk space for installation.

## “Cannot run program ‘…’ (in directory X) error=2 No such file or directory” [unresolved]

* (re)installed git and ran the script that came with it to add the install to my environment variables/path
* Made sure JDK path correct