Tools of efficiency – Creating a more streamlined programming experience

In the realm of programming, projects are both commonplace and imperative, and they can also be one of the most tedious, time consuming, and stressful tasks. It is guaranteed that every project will each have their own set of stipulations and challenges, and depending on the scale of the project, one may want to use a certain range of tools or techniques that can help to complete it satisfactory – as to reap the benefits of alleviating stress and increasing workflow. In order to help understand the necessity of said tools and techniques, some of the most routine challenges that come with working on medium to large sized projects are the following:

- Maintaining a relatively large file base.
- Verifying that all code is correct (by fulfilling all purposes of their intent).
- Maintaining a non-trivial number of classes, methods, or functions (20+).

The particular solutions that address these common challenges listed above in respective order are the following: version control, unit testing, and debugging tools. The scope of the utility of these reaches far beyond a single project, however. These are tools that can be used career-long, and provide an overall more streamlined programming experience. From this point forth, we will take an in-depth look at these techniques and tools from an objective standpoint.

Version control: The basics

In the beginning of every programmer's journey, one might start by creating a simple program that outputs "Hello Word." The next step further might be to write a simple function that adds or subtracts numbers. Beyond this level of complexity, programs might encapsulate multiple files, which might also include different file types as well. At this point, a widely used tool known as version control should be considered. Version control, in its most basic sense, is a way for programmers to store their files somewhere other than their local machine. This is typically done by having files stored on the database of some website. The utility of this might seem obvious – and it is, or at least partially. Having backup copies of important files is indeed a good idea, but version control is able to provide much more than just that.

An example: GitHub

One of the most popular choices for version control is GitHub, which contains all features of a version control system that were previously described. It works by downloading a piece of software called Git onto your local machine, and using that piece of software through either a UNIX terminal or their own shell (for non-UNIX systems) to interact with the GitHub website.

GitHub: The pros

In addition to storing backup copies of files, version control can allow you to easily collaborate with others on a project, release downloadable versions of a project, see your history of changes, or even revert back to a previous version of a project. If you have ever lost or corrupted an important file, worked in a group where a file was needed to be passed around, or made multiple copies of a file to differentiate between versions, or would like to utilize one of the features described above, then version control is a solution that will save many organizational headaches down the road.

The benefits: collaborative work using GitHub

Collaboration is one of the necessary components of software engineering, and there's no getting around it. To be successful, all partners of a team must have two things: good communication skills and a GitHub account. On a high level, GitHub facilitates group work by making it easy for all members to be able to:

- Update their files by getting the most recent version from the website.
- Make changes to current files – even if someone else is already doing the same.
- Release specific versions of a project

GitHub: A 10 step tutorial on setting up a personal repository

1) Install Git: For UNIX users on a Debian distribution like Ubuntu, open up a terminal and use the command: **sudo apt-get install git-all**. For Windows users, navigating to the site http://git-scm.com/download/win will automatically initiate the download. Follow any further instructions given by these downloads in order to install.

2) Go to http://www.github.com and create an account.

3) Once you have created account, log in and click the green button that says "New repository". This will eventually be a folder that will be able to be accessed on your local machine later on.

4) Fill out the information such as the repository name and description. Checking the box that says "Initialize this repository with a README" is useful for giving detailed project descriptions, but not typically for individual folders.

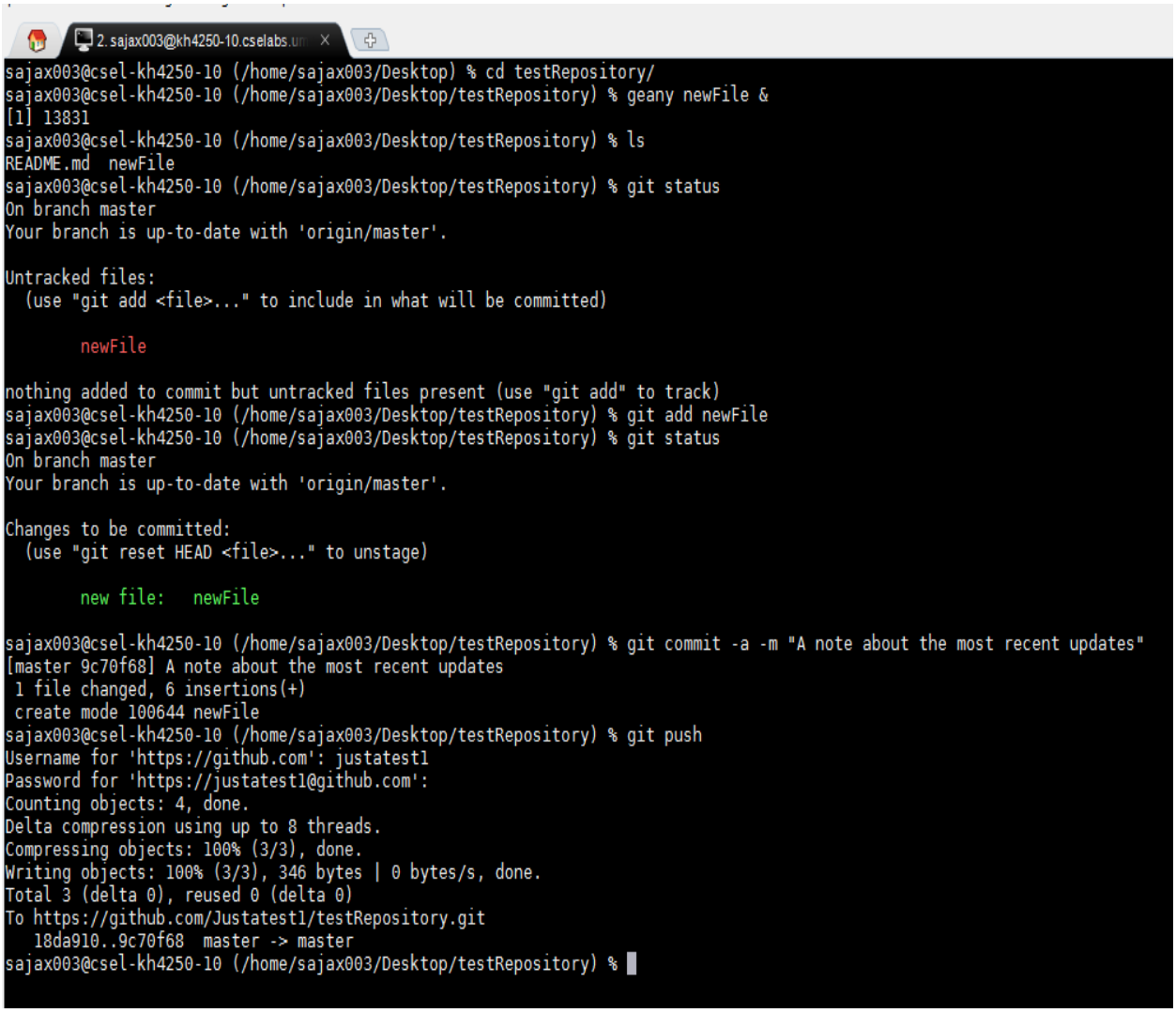
5) After this has been done, GitHub will redirect to a page that might feel unfamiliar. This page is simply the homepage for the repository that was just created.

6) Locate the clipboard button and copy the contents of the address. This is used to give your local machine information about where the repository is on the web.

7) Open up a terminal or the Git Shell. Traverse to the directory where you would like to set up the repository.

8) Through the command "git clone <address of repository>", you will be able to take the repository you made on the web and bring it down to your local machine. The address of the repository is what you had copied from the clipboard earlier.

9) Add a test file to the local repository and attempt to push it to the repository – The screenshot below shows how this can be done.



10) You have now successfully synchronized a repository to be used with GitHub!

GitHub: A final note

Many vital Git commands have been somewhat glossed over in step 9. I have created a helpful list of commands (including all used in step 9) with information explaining what each one does in the picture below. Each command will start with the keyword “git,” followed by the commands listed in bold.

Git Cheat Sheet

- **config**: Used to configure various settings of Git (i.e: email, name, filemode, editor). These settings can be checked by calling the command: git config -l
- **clone**: One of the ways you can get access to a public repository on a local machine is to clone it, which will create a copy of a repository on your local machine. Its placed in the current directory you're working in. This is used in the form: git clone "Repository URL here".
- **remote**: Used to see which repositories you have configured. Can also be specified with the -v identifier to show the URL's that git has stored.
- **status**: Used to check the status of the files on Github compared to your local ones, (i.e: to see if any files are untracked, or still need to be committed).
- **commit**: After editing and saving a file locally, you use this command to prepare your changes to be pushed to Github.
- **push**: After editing, saving, and committing a file locally, use this to update the given file on the respective repository on Github.
- **pull**: If changes to a file were made via Github, use this to update any changes to a local file.
- **add**: If a file was created locally, it will naturally be untracked. Use this command to track it (it will then be 'pushed' to the proper repository on Github).

The cons

Though GitHub and Git are great tools that every programmer should use, not every tool is perfect, and these come with certain drawbacks. One problem with Git is that it requires a high level of communication between groupmates if it's being used for team collaboration. Someone could alter a file and push it to the repository, while another person does the same, though it would tell the person pushing that they first need to pull. Also, without properly formatting the message of the commit every time a piece of code needs to be pushed, another groupmate could pull without knowing that some of the pushed code could adversely affect the entire project.

Unit testing / Test Driven Development

One useful tool that is often swept under the rug during development is unit testing. This technique is implemented by calling a function with an input against some expected output in order to see if it will return true or false, as to give the programmer an indication whether a specific method or function works. This is **not** simply implemented in the specific language being worked with. Moreover, there are things called frameworks (akin to a code library) which accomplish this.

A case for practicing unit testing

The major reason unit testing is used is to make sure that every function is correctly fulfilling all aspects of its intention/purpose. Should this not be the case, adding an extra function or method to a program may cause undesired behavior, even if it is itself correct. Once all "units," have been tested, then larger parts of the program can be tested in conjunction and also contain unit tests. Another reason that explains its popularity is that it has utility for group work and collaboration. As all tests in a framework

are relatively standardized, so a test from one programmer should not vary greatly from another programmer, whereas their code for writing the same function or method may look completely different.

In short, routine unit testing will save many headaches, and quite possibly a significant amount of time for projects of a larger scale. For simple projects, it may be arguably more efficient to skip this practice, but any medium or large scale project that needs to be maintained will almost always use unit testing.

CxxTest: A testing framework for C++

CxxTest is a free unit testing framework for C++ code, and it comes with many basic functionalities that you might see in other frameworks. One useful keyword to know for this specific framework is “test suite.” A test suite is simply an aggregate of test cases defined in the same header file. An example of the syntax of the tests is as follows:

```
#include <cxxtest/TestSuite.h>

using namespace std ;

class WordCountTestSuite : public CxxTest::TestSuite
{
public:

    // Some simple sample tests to illustrate the assertion macros.
    void test_addition_sample () {
        TS_ASSERT ( 1 + 2 == 3 ) ;
    }
    void test_addition_equality_sample () {
        TS_ASSERT_EQUALS ( 1 + 2, 3 ) ;
    }
}
```

In this case, the expected output of the tests are both true. Unfortunately, one of the biggest drawbacks of this framework is the amount of setup it takes to enable these tests to run, as the process is fairly involved for a newcomer to unit testing. Information that explains how tests are able to be run can be found on http://cxxtest.com/guide.html#gettingStarted

Test Driven Development

The practice of test driven development refers to writing the tests for a piece of code before the code itself is written. This is usually done for the purpose of being able to have an inkling of how functions might work, and which ones are essential for the project. This approach is also helpful because if one cannot even think of tests to write, there’s a fairly good chance that the actual code either won’t be written at all, or be fairly difficult to produce.

GDB: A tool to use in conjunction with Unit testing

One of the most common words in a C/C++ programmer’s vocabulary is, “segmentation fault.” From the inexperienced to the seasoned veterans, it is safe to say that nearly everybody has encountered this

message at some point in their programming career. One of the most peculiar aspects of a message such as this is that it gives the programmer no indication of where an error in a program might have occurred. Depending on the size and type of the program, this could mean either a simple two minute fix, or an impending nightmare of looking through hundreds of functions whilst trying to use print statements to locate the error. As a general rule of thumb, if there is any uncertainty about why some piece of code is behaving in a certain way, one possible solution would be to use a tool called GDB.

GDB: A brief overview

GDB is considered a traditional debugger. The most succinct summary of GDB can be found on the GDB page of the GNU website, “GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.” (www.gnu.org)

More specifically, this allows you to find the specific point in your program where something went awry. Additionally, you can view the values of variables alongside the program execution. For a novice programmer, these would be the most pertinent concerns regarding using GDB.

GDB: A simple example

In order to give a more clear understanding of how this tool works, I have created a contrived example of C++ code that has a few things wrong with it.

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
using namespace std;

int main() {
    int *ptr;
    int foo[3] = {0, 1, 2};

    for (int i = 0; i < 5; i++) {
        *ptr = foo[i];
        cout << "Value of ptr is " << *ptr << endl;
    }
}
```

```
sajax003@csel-kh1200-05 (/home/sajax003/Desktop) % g++ -g -c test2.cpp
sajax003@csel-kh1200-05 (/home/sajax003/Desktop) % g++ -g -o test2 test2.o
sajax003@csel-kh1200-05 (/home/sajax003/Desktop) % ./test2
Segmentation fault (core dumped)
sajax003@csel-kh1200-05 (/home/sajax003/Desktop) %
```

The utility of having GDB is that we now have a tool that might be able to tell us where exactly this program had a segmentation fault. In order to run GDB, you can first check if it’s installed by running the following command in a terminal: gdb –help. If it is not installed, it can be downloaded at http://www.ftp.gnu.org/gnu/gdb . Once installed, the debugger can be invoked by simply calling “gdb”

from the command line followed by an optional program name afterward. The following gdb session shows how the line at which the program above had a segmentation fault was deduced.

```
sajax003@csel-kh1200-05 (/home/sajax003/Desktop) % gdb test2
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from test2...done.
(gdb) break 10
Breakpoint 1 at 0x400830: file test2.cpp, line 10.
(gdb) run
Starting program: /home/sajax003/Desktop/test2
warning: File "/soft/gcc-4.5.2/ubuntuamd1/lib64/libstdc++.so.6.0.14-gdb.py" auto-loading has been declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
        add-auto-load-safe-path /soft/gcc-4.5.2/ubuntuamd1/lib64/libstdc++.so.6.0.14-gdb.py
line to your configuration file "/home/sajax003/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/sajax003/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
        info "(gdb)Auto-loading safe path"

Breakpoint 1, main () at test2.cpp:10
10              for (int i = 0; i < 5; i++) {
(gdb) print foo[0]
$1 = 0
(gdb) print foo[1]
$2 = 1
(gdb) print foo[2]
$3 = 2
(gdb) next
11                      *ptr = foo[i];
(gdb) print *ptr
$4 = -1992206527
(gdb) next

Program received signal SIGSEGV, Segmentation fault.
0x0000000000400846 in main () at test2.cpp:11
11                      *ptr = foo[i];
```

A forewarning

Do not be misled. GDB does not necessarily help fix your code, and you still might end up getting results that do not result in much debugging progress at all. It is mainly used to spot errors in code. More importantly, and perhaps most obviously, it is good to know about what is considered legal in the specific language that is being worked with. For example, C++ does not implement bounds-checking on arrays, and as a result, it is possible to write programs that have seemingly correct behavior when they are indeed incorrect, or vice versa. Thus, it is a good idea to consult GDB for cases of unintended program behavior.

In Summary

Whether or not these tools and techniques are actually utilized in a project, they still give some insight into the role of outside utilities in the software development process. Version control imperative for a programmer's organization as they increase their collection of files – project related or not. It's excellent for collaborating with a team and staying organized. Additionally, unit testing is industry standard and provides a good way for standardizing tests, no matter the amount of people assigned to work on such tests. GDB is also important because we will encounter many segmentation faults in our journey as programmers.

Works Cited

"Downloading Git." *Git*. N.p., n.d. Web. 16 Dec. 2015. <http://git-scm.com/download/win>.

Www.github.com. N.p., n.d. Web. 13 Dec. 2015. <https://github.com/>.

Saja, John. “Mobaxterm session.” n.d. Author’s Screenshot. 13 December 2015.

Saja, John R. *GitHub*. N.p., 25 Jan. 2015. Web. 15 Dec. 2015. <https://github.umn.edu/umn-csci-2041S15/repo-sajax003/blob/master/cheat-sheet.md>.

Saja, John. “CxxTest code.” n.d. Author’s Screenshot. 13 December 2015.

"CxxTest User Guide." *CxxTest User Guide*. N.p., n.d. Web. 13 Dec. 2015. <http://cxxtest.com/guide.html#gettingStarted>.

"GDB: The GNU Project Debugger." *GDB: The GNU Project Debugger*. N.p., n.d. Web. 6 Dec. 2015. <https://www.gnu.org/s/gdb/>.

Saja, John. “GDB example code.” n.d. Author’s Screenshot. 13 December 2015.

Saja, John. “Terminal core dump.” n.d. Author’s Screenshot. 13 December 2015.

N.p., n.d. Web. 15 Dec. 2015. <http://www.ftp.gnu.org/gnu/gdb>.

Saja, John. “GDB session.” n.d. Author’s Screenshot. 13 December 2015.