**DevNet Hands On Session - Week 5**

* TDD: Test Driven Development
  + advantage:
    - High quality of code - think of the product rather than implementation of the product
    - Early detection of bugs
    - Make sure one development does not break other tests
  + example:
    - textfsm development
      * We need to make sure any additions/modifications to the template does not break the template for previous versions of output
  + 3 steps:
    - Write tests that fail
    - Write code that makes test pass
    - Run tests that pass
  + the tests should never be modified
    - if the architecture of product changes in future, make new set of tests
* Unittest:
  + testing smallest individual component of the code
  + helps narrow the cause of bugs
  + Integration test: testing interoperability between components
* Python Unittest:
  + Developing Calulator app with just addition and Division with TDD approach
  + Other testing frameworks in python:
    - pytest
    - Nose
  + import module unittest
  + inherit class from unittest.TestCase
  + test methods should start with 'test\_\*' for auto-discovery
  + one test method should only test one functionality
  + assertEqual, assertRaises, context manager for handling exceptions ex: with self.asserRaises(ValueError):
  + setUp and tearDown methods for test setup and test teardown
  + classmethod setUpClass and tearDownClass methods for suite setup and teardown
  + tests are run in random order

If getting error for importing module

# \* For Mac

export PYTHONPATH="${PYTHONPATH}:/path/to/your/project/"

# \* For Windows

set PYTHONPATH=%PYTHONPATH%;C:\path\to\your\project\

**BASH**

Online Practice Link: <https://repl.it/languages/bash>

WSL Resources : <https://docs.microsoft.com/en-us/windows/wsl/>

Ubuntu on WSL : <https://ubuntu.com/wsl>

All Commands Reference Link : <https://ss64.com/bash/>

**Practice Commands**

man man

sudo apt-get update

cat weather.py | more

|  |  |
| --- | --- |
| **$ cd /** | Changes directory to the root directory |
| **$ cd /home/username** | Changes directory to the /home/username directory |
| **$ cd test** | Changes directory to the test folder |
| **$ cd ..** | Moves up one directory |

|  |  |
| --- | --- |
| **$ pwd** | Print your current working directory |
| **$ ls** | Lists files and directories in the current working directory |
| **$ ls -a** | Lists everything in the current directory, including hidden files |
| **$ ls /home/username** | Lists everything in the /home/username directory |
| **$ ls -l** | Lists permissions and user and group ownership |
| **$ ls -F** | Displays files and directories and denotes which are which |

|  |  |
| --- | --- |
| **$ mkdir test** | Makes a new directory called test in the current working directory if you have permission |
| **$ mkdir /home/username/test** |  |
| **$ cp sydney.txt sydney2.txt** | Copies a file called sydney.txt from the current directory and names the copy sydney2.txt |
| **$ cp /home/username/sydney.txt ~/sydney2.txt** | Copies a file as described above but using the full path and the home directory path |
| **$ cp -r folder folder.old** | Copies a folder |

|  |  |
| --- | --- |
| **$ mv caleb.txt calebfinal.txt** | Renames a file called caleb.txt to calebfinal.txt |
| **$ mv /home/username/caleb.txt ~/calebfinal.txt** | Renames a file as described above but using full paths |
| **$ mv -i \* /home/username/new/** | Moves all files and directories in the current folder to a directory called new |
| **$ rm test.txt** | Deletes the file test.txt in the current working directory |
| **$ rm -rf test** | Forces the deletion of the folder test and everything in it |
| **$ touch emptyfile.txt** | Creates an empty file named emptyfile.txt |
| **$ touch file{1..20}.txt** | Bulk creates files from file1.txt to file20.txt |
| **$cat file1.txt** | Displays the contents of file1.txt |
| **$cat file1.txt | more** | Displays the contents of file1.txt and pipes the output to more to add page breaks |
| **$cat >file2.txt** | Sends a user’s typed or copied content from the command line to fil |
| **$env | more** | Shows all environment variables with page breaks |

$ echo $PATH

$ echo "export PATH=$PATH:/🡨path--> " >> ~/.bashrc

source ~/.bashrc and . ~/.bashrc give equivalent result.

**CONDUCTING CODE REVIEW**

Key benefits of code review are:

* Better code quality  – improve internal code quality and maintainability (readability, uniformity, understandability, ...)
* Finding defects  at very early stage – help find not only logical problems but also performance problems, security vulnerabilities, injected malware, ...
* Knowledge transfer  – help in transferring knowledge and enable learning process in the Agile software development team
* Increase sense of mutual responsibility  – increase a sense of collective code ownership and solidarity, which fits into the key Agile principle
* Suggesting better solutions  – generate ideas for new and better solutions during the code review and discussion.
* Ensuring QA guidelines compliance – Code reviews can be made mandatory for some industry or some critical code

In Agile software development where development pace is fast, code review tools become more and more important as it help automate the review process to minimize the reviewing task of the code.

**Gerrit**

This is an open-source tool originating out of Google, it is built on top of the “Git version control system”. Gerrit is a lightweight solution to pre-acceptance (codebase) code review. It is exceptionally useful in project environments where all all changes.



Gerrit is at once a “changes staging area” where they can be reviewed before being accepted into the codebase inn source control. The tool is widely used in Agile software development environment.

**Key Features of Gerrit are:**

* side-by-side difference viewing and inline commenting
* Integrated with Git version control system allowing authorized contributors to merge Changes to the Git repository, after reviews are done.
* Simple review workflow: contributor uploads Changes to Gerrit with Git, peers use the web browser to make reviews

  Link to download: <https://www.gerritcodereview.com/>

Following are some good practices to help make your code review effective:

* Use a code review checklist that includes organization-specific practices (naming conventions, security, class structures, and so on) and any areas that need special consideration. The goal is to have a repeatable process that is followed by everyone.
* Review the code, not the person who wrote it. Avoid being robotic and harsh so you don’t hurt people’s feeling and discourage them. The goal is better code, not disgruntled employees.
* Keep in mind that code review is a gift. No one is calling your baby ugly. Check your ego at the door and listen; the feedback you receive will make you a better coder in the long run.
* Make sure the changes recommended are committed back into the code base. You should also share findings back to the organization so that everyone can learn from mistakes and improve their techniques.