**INTRODUCTION TO PROGRAMMING**
**WEEK 1 - THE UNIX OS AND RUNNING PYTHON SCRIPTS**

**Before using these notes, please first complete the relevent course pre-work.*

**UNIX OS**

The Unix OS is important for us because it allows us to run python scripts off the corp network to affect files, directories, and push applications onto web-servers.

In reality all this can also be done off of your personal laptop, but there are several key advantages to running it off of the corp network:

- Accessibility (ability to affect personal/shared filers, versus just your computer's hard drive).
- Sharing (others can run your scripts and you can utilize scripts written by others)
- Version control (maintains standard copies of scripts for teams/groups to use).

Engineers almost always use Unix for their day to day work, and in order for us to interact with the corp network, we will also need to do the same.

While much of this might seem unnecessary, it will all pay off in the end and it will give you a good baseline foundation for understanding for how engineering is done at Google.

**GETTING STARTED**

In order to utilize the Unix OS on the corp network, we first need to connect to a computer that is actually enabled with Unix OS. You will often hear engineers refer to this as their "linux box", "box", or "machine". They all pretty much mean the same thing - a Goobuntu desktop that is constantly connected to the corp network. Your computer is probably a PC or Mac so it can't do the the trick, so instead we will connect to my machine whose name is **gswitch.mtv.corp.google.com** or **gswitch** for short.

**PC:**

1. Click on `Start` **>** `All Programs` **>** `Google Stuff` **>** `Putty`
    a. If you are on Windows 7, you will need to install Putty first.
2. Drag the icon onto your desktop so that you have a shortcut
3. Double click the `Putty` icon to begin
4. Under `Saved Sessions` type: `gswitch.mtv.corp.google.com`
5. Click `Save` and `Load`
6. Copy and paste `gswitch.mtv.corp.google.com` into the `Host Name` text field at the top of the window.
7. Click `Open` to begin
8. Log in with your corp LDAP/Password
    a. When typing, the words won't appear, but still hit enter after typing it in

**Mac**

1. Open up a terminal window
2. Type: `ssh gswitch`
    a. If you are not in MTV, this will need to be: `ssh gswitch.mtv.corp.google.com`
3. Log in with your corp LDAP/Password

Try typing the command:

```
ls -l
```

What do you notice? Where are you now?

## BASIC COMMANDS

In this section, you will learn basic commands that allow you to navigate and manipulate files/ folders from the command line. All of this can be done through the finder (mac) or windows explorer (windows). The reason you need to learn to use the command line is because python scripts are executed through the command-line (and in the long-run it's more powerful/ convenient than using OS navigators).

From here on out, a "folder" will be known as a "directory".

### Listing the contents of a directory

The first command you used above was `ls`. ls stands for "list" and it is a command that will display the contents of a directory (files and sub-directories). To use it, just type the following into the command line:

```
ls
```

A variation of this command is passing the `-l` flag to it which indicates that you would like to see a "long" listing. This lists everything in the directory and but also shows details for each file/ directory.

```
ls -l
```

### Creating a Directory

To create a directory, invoke the `mkdir` command. This stands for "Make Directory". Simply invoke the command along with the name of the new directory.

```
mkdir new_directory
```

**Removing a Empty Directory**

To remove an empty directory, invoke the `rmdir` command. This stands for “Remove Directory”. Simply invoke the command along with the name of the target directory to be deleted.

```
rmdir new_directory
```

**Changing Directories**

To “go into” a directory or “enter” it, invoke the `cd` command. This stands for “Change Directory”. Simple invoke the command along with the name of the target directory you want to enter.

```
cd new_directory
```

To enter sub-directories, simply separate the hierarchies with forward slashes. For instance, if `new_directory` had a sub-directory called `sub_dir_1` and `sub_dir_1` had a sub-directory called `sub_dir_2` you could navigate directly there with this command:

```
cd new_directory/sub_dir_1/sub_dir_2
```

To go backwards and go into a parent directory, use the `..` qualifier. For instance, the following commands will take you to the `sub_dir_1` level.

```
cd new_directory/sub_dir_1/sub_dir_2
cd ..
```

As always, the `..` qualifier can be separated by forward slashes to go back multiple levels. The following will take you to `new_directory`.

```
cd new_directory/sub_dir_1/sub_dir_2
cd ../..
```

One shortcut for changing directories back to your home directory is to use the `~/` qualifier

```
cd ~/
```

This will take you back to your home filer.

**Creating Files**

To create a file, invoke the `touch` command along with the name of the file you'd like to create:

```
touch test.py
```

**Viewing Files**

To view a file, invoke the `cat` command along with the name of the file you'd like to create:

```
cat test.py
```

This will show the contents of the file inside your terminal window. If the file is blank, it won't show anything at all.

**Editing Files (Optional)**

For our purposes, you can just use your OS's navigator (finder or windows explorer) to find the file and right click and edit with IDLE or TextWrangler. However, if you are curious about using command-line editors, I would recommend `vi`.

- **vi <file name>** - edit the target file
- **esc (hit the "esc" button) + i** - insert, edit the text of a file
- **esc + :q!** - quit editing a file without saving
- **esc + :wq** - quit and save the file you were editing

**Deleting Files**

To delete a file, invoke the `rm` command along with the name of the file(s).

```
rm test.py
```

If you had created multiple files you want to delete, just invoke and separate filenames with spaces.

```
rm test.py test2.py
```

**Deleting Directories with Content**

If you want to delete a directory and it has stuff in it, you won't be able to use `rmdir`. Instead, invoke rm along with the `-r` flag (recursive). This will delete everything in the directory along with any children of the entities inside the directory.

```
rm -r new_directory
```

**Changing Permissions**

Changing permissions on a file or directory is an extremely effective way to protect it. When you invoke `ls -l` on a directory's contents you will notice that entities are listed like this:

```
-rwxr-xr-x 1 alberthwang google 259 Jul 21 15:13 comment_test.py
```

The first part of this listing is a permissions signature:

```
-rwxr-xr-x
```

This looks like some gibberish but it will all make sense in a second. All permissions take this form:

```
_ | _ _ _ | _ _ _ | _ _ _
```

The pipes don't actually show up but serve to demarcate the various sections of the signature:

```
entity type | self permissions | group permissions | others permissions
```

In the example above, It would look like this:

| Signature | Entity type | Self Permissions | Group Permissions | Others Permissions |
|---|---|---|---|---|
| -rwxr-xr-x | - | rwx | r-x | r-x |

In the first section, entity type, we will only be concerned with 2 types, files and directories. Files are specified with a dash, -, and directories with a d.

```
-rwxr-xr-x (this is a file)
drwxr-xr-x (this is a directory)
```

The next sections specify permissions on the file for various user groups. These permissions can include any of 3 abilities:

```
r - read (the ability to open the file and look at contents)
w - write (the ability to change the contents of the file)
x - execute (the ability to run the script)
```

What does this signature tell us?

```
-rwxr-xr-x
```

This tells us that:

- the creator of the file (`self`) has read/write/execute (`rwx`) permissions
- the creator's group (`group`) has read/execute (`r-x`) permissions
- people outside the creator's group (`others`) have read/execute (`r-x`) permissions.

`-rwxr-xr-x`

To change the permissions on a file or directory, invoke the `chmod` (change mode) command like so:

```
chmod <numerical permissions> <filename>

chmod 765 test.py
```

The numerical permissions is a 3 digit number representing the permissions you want to allocate to the file/directory. Each digit represents a different group (`self` - 1st digit, `group` - 2nd digit, `other` - 3rd digit). The number is calculated using this mapping (and adding together the permissions you want):

```
4 = read
2 = write
1 = execute
```

If you wanted to give a group read/write/execute (`rwx`) permissions, the numerical representation of that permission would be `7 (4 + 2 + 1)`. Let's take a look at the example chmod command again:

```
chmod 765 test.py
```

This command tells the system tot:

- give the creator of the file (`self`) has read/write/execute permissions 7 = 4 + 2 + 1
- give the creator's group (`group`) has read/write permissions 6 = 4 + 2
- give people outside the creator's group (`others`) read/execute permissions 5 = 4 + 1

Let's do a few to practice...Represent these as permission signatures (`-rwxr-xr-x`):

```
chmod 762 test.py # -rwxrw--w-
chmod 700 test.py # -rwx------
chmod 734 test.py # -rwx-wxr--

mkdir test_dir
chmod 760 test_dir # drwxrw----
```

Here are the answers:

```
-rwxrw--w-
-rwx------
-rwx-wxr--
```

## SO WHY THE COMMAND LINE?

Many of you are probably thinking - so why do this at all? Why not just manipulate the files through a viewer of some sort (like you probably normally do).

It's true, working with the corp network through a file viewer is definitely easier (at first) and we will be doing a lot of that in this class as well. However, using the command line has several key advantages in programming:

1. Ability to invoke scripts and run programs.
2. Ability to run scripts and specify targets (instead of just running a script in a singular location, as you would through some sort of software on your hard drive).
3. Ability to batch commands together to execute shell scripts (sort of like batch files for those familiar with MS-DOS).
4. Eventually, with practice - it is faster to access/edit files through the command line than through the file viewer.
5. It is a standard practice in software engineering (especially at Google).

## RUNNING A PYTHON SCRIPT

So let's get to it.

1. shell into `gswitch` (putty or terminal/ssh)
2. create a directory in your home filer called `python`
3. create a directory in `python` called `week1`
4. create a file in `week1` called `helloworld.py`
5. open up your filer in the a file viewer (how you usually do it)
6. navigate to the `helloworld.py` file and open with the python file editor (IDLE or TextWrangler)
7. type (do not copy/paste):

```
#!/usr/bin/python2.6

print 'hello world!'
```

8. First `cd` into the correct directory and then in your terminal window run the python script like so:
   - `python helloworld.py`

9. now try running the program from a level up, or two levels up
10. now try altering the script to print `hello friend!`
11. now try renaming the file to `hellofriend.py` and running it (all from command line)
12. now try editing the file to make it say `what is up, friend?!`

## APPENDIX A - QUICK REFERENCE FOR BASIC UNIX COMMANDS

**ls** - lists all files in a directory
**ls -l** - lists all files in a directory but with permissions, owners, and groups
**cd <directory name>**- change directory

- **cd ~/** - change back to the home directory
- **cd ..** - change to a directory above
- **cd ../..** - change directory to 2 levels above (you get the idea)

**mkdir <directory name>**- make directory
**rmdir <directory name>**- remove directory without content
**touch <file name>** - create a file with that file name
**rm <file name>** - remove the target file
**rm -r <directory name>** - remove the target directory with content in it
**vi <file name>** - edit the target file
**esc + i** - insert, edit the text of a file
**esc + :q!** - quit editing a file without saving
**esc + :wq** - quit and save the file you were editing
**mv <file1> <file2>** - moves content of file1 into file2 and gets rid of file1

- If file2 does not exist, creates file2
- Use this to rename files and directories

**cp <target location> <new location>** - copies target file into new location
**cp -r <target location> <new location>** - copies target files in directory to new directory
**chmod <numerical permissions> <filename>** - change permissions

- reading permissions: _ | _ _ _ | _ _ _ | _ _ _
- example: -rwxr-xr-x
    - - | rwx| r-x | r-x
    - file, (read/write/execute) owner, (read/execute) group, (read/execute) others
- 4 = read, 2 = write, 1 = execute (7 = 4 + 2 + 1 = read/write/execute)
- each number corresponds to a position in the permissions path (7|5|5)
- example: chmod 755 test.txt

## APPENDIX B - MORE ABOUT GROUP PERMISSIONS

The group that we refer to in the chmod and permissions signature section is a cdb (corpDB group). We are all a part of several CDB groups. You can see all the groups you are in (join groups, etc.) by going to groupman:

http://groupman/

You can also see your CDB settings through the command line:

```
cdb user show <ldap>
```

From the command-line you can see your primary group (marked with an asterisk). When you create files/directories, the group assigned to those newly create entities is your primary group.

You can change the group on any file you own with the `chgrp` command:

```
chgrp <groupname> <entity>
```