INTRODUCTION TO UNIX

ERIC MARTIN

1. Generalities

Using the *Terminal* application, open an *x-term window*. You type your commands in an x-term window.

- Many commands take one or more *arguments*.
- Many commands can take one or more *options*.
- The *short* options start with one hyphen (`-`) followed by one letter, and sometimes an argument for the option.
- The *long* options start with one or two hyphens (`-` or `--`) followed by a string (word or sequence of words usually separated with a hyphen), and sometimes an argument for the option.
- Many short options that do not take an argument can be combined together, with one hyphen followed by the letters of the options.
- Many arguments are *optional*.

2. A few Unix commands

Try the following commands.

- `date`.
- `cal` (no argument), `cal 2018` (one argument), `cal 2 2018` (two arguments).
- `pwd` to **p**rint the **w**orking **d**irectory, which after you have logged in and before you have done anything, is also your *home directory*.
- `ls` to **l**i**s**t the *files* (of which *directories* are a particular case) in the working directory, excluding the *hidden files* whose name starts with a dot (these are usually configuration files that are seldom modified or read).
- `ls -A` (one short option) to list **a**ll files in the working directory.
- `ls -l` to get a **l**ong listing of the files in the working directory, excluding the hidden files.
  - The most common characters for the first character are `-` for a regular file, and `d` for a directory.
  - The next three characters indicate whether the file is *readable* (`r`) or not (`-`), *writable* (`w`) or not (`-`), and *executable* (`x`) or not (`-`) by the *owner* of the file.
  - The next three characters provide the same information for the users who belong to the same *group* as the owner of the file.
  - The next three characters provide the same information for all other users.
- `ls -l -A` or `ls -a -l` or `ls -lA` or `ls -al` to use `ls` with both short options.
- `mkdir` followed by a number of directory names to **ma**k**e** (create) some **dir**ectories. The names can be either
  - *absolute paths*, that start with the string that `pwd` returns when it is executed in your home directory;
  - paths that are *implicitly relative* to the working directory;
  - paths that are *explicitly relative* to the working directory, starting with `./`;
  - paths that are explicitly relative to the *parent* of the working directory, starting with `../`. More generally, `..` can be used in paths to go one level higher in the *hierarchy* of directories;
  - paths that are explicitly relative to your home directory, starting with (`~/`).

  For instance, assume that `pwd`, when executed in your home directory, prints out

  `/import/kamen/1/pythonist235`

  So your user name is `pythonist235`. Assume that your home directory contains a subdirectory named `COMP9021` which itself contains a subdirectory named `Labs`. Finally, assume that your working directory is the subdirectory `COMP9021` of your home directory. So `pwd`, executed in this working directory, outputs

  `/import/kamen/1/pythonist235/COMP9021`

Date: Session 1, 2018.

Now assume that you want to create the subdirectories `Quizzes`, `Assignments` and `Lectures` of the subdirectory `COMP9021` of your home directory, a subdirectory `COMP9311` of the home directory, and a subdirectory `Lectures` of the directory `COMP9311`. Then corresponding to the 5 options listed above, you could execute:
  - `mkdir /import/kamen/1/pythonist235/COMP9021/Quizzes`
  - `mkdir Assignments`
  - `mkdir ./../COMP9311`
  - `mkdir ../COMP9021/Lectures`
  - `mkdir ~/COMP9311/Lectures`

Of course, rather than the first, third and fourth commands above, it would be more natural and effective to execute instead:
  - `mkdir Quizzes`
  - `mkdir ../COMP9311` or `mkdir ~/COMP9311`
  - `mkdir Lectures`

- `touch` followed by a number of file names to create empty files or to modify the *last modification date* of existing files. When creating empty files, `>` is a simpler alternative to `touch`. For instance, to create two files *`file_name1`* and *`file_name2`* in the working directory you can type either `touch` *`file_name1 file_name2`* or `>`*`file_name1`* `>`*`file_name2`* (with or without spaces after `>`).
- `cd` to **c**hange (go to another) **d**irectory. This command can be followed by:
  - no argument, in which case the new directory is the home directory;
  - an absolute path name;
  - a pathname that is implicitly relative to the working directory;
  - a pathname that, starting with `./`, is explicitly relative to the working directory;
  - a pathname that, starting with `../`, is explicitly relative to the parent of the working directory.
  - a pathname that, starting with `~/`, is explicitly relative to your home directory.

  For instance, assume that your working directory is `~/COMP9021/Labs`, that is, the subdirectory `Labs` of the subdirectory `COMP9021` of your home directory. Also assume that you first want to go to your home directory, and from there to the directory `~/COMP9021`, and from there to `~/COMP9021/Quizzes`, and from there to `~/COMP9311/Lectures`, and from there to `~/COMP9021/Lectures`, and from there to `~/COMP9021`. Then corresponding to the 6 options listed above, you could execute:
  - `cd`
  - `cd /import/kamen/1/pythonist235/COMP9021`
  - `cd Quizzes`
  - `cd ./../COMP9311/Lectures`
  - `cd ../COMP9021/Lectures`
  - `cd ~/COMP9021`

  Of course, rather than the second, fourth and sixth commands above, it would be more natural and effective to execute instead:
  - `cd COMP9021`
  - `cd ../COMP9311/Lectures` or `cd ~/COMP9311/Lectures`
  - `cd ..`
- `mv` *`file_name directory_name`* to move the file *`file_name`* to the directory *`directory_name`*, where *`file_name`* and *`directory_name`* can be either relative or absolute paths.
- `cp` *`file_name1 file_name2`* to copy the file *`file_name1`* and give it the name *`file_name2`*, where *`file_name1`* and *`file_name2`* can be either relative or absolute paths.
- `cp` *`file_name directory_name`* to copy the file *`file_name`* in the directory *`directory_name`*, where *`file_name`* and *`directory_name`* can be either relative or absolute paths.
- `rmdir` followed with some directory paths to **rem**ove those **dir**ectories, provided that they are *empty*, *i.e.*, do not contain any file.
- `rm` followed with some regular file paths to **rem**ove those files.
- `rm -r` followed with some directory paths, *i.e.*, the previous command provided with one short option and directory paths as arguments, to **r**ecursively remove those directories and everything they contain, down to any depth. To be used with utmost care...

Command completion is a useful feature of the *bash shell*, the command-line interpretation we are using. By pressing the **tab** key, you let bash complete what you are typing. For instance, suppose that you want to go from the working directory to a subdirectory whose name starts with `COMP`. Suppose that you type `cd COMP`.

- If only one subdirectory has a name that starts with `COMP`, say `COMP9021`, then pressing the tab key after `cd COMP` automatically completes the command to `cd COMP9021`.

- If no subdirectory has a name that starts with `COMP`, then pressing the tab key again and again after `cd COMP` will just make your computer beep, or flash, or complain in some way.
- If many subdirectories have a name that starts with `COMP`, then pressing the tab key once after `cd COMP` will make your computer complain, but pressing the tab key a second time will display the list of all subdirectories whose name starts with `COMP`, and let display `cd` with its incomplete argument again, giving you hints on how to complete it partially or totally.

You can also use the uparrow and the downarrow of your keyboard to retrieve commands you have typed previously.

2.1. **The chmod command.** Recall from previous section what the `ls -l` command outputs. When you want to change the permissions of some file, you use the `chmod` command to **ch**ange the **mod**e of the file.

- With the options, `+r`, `+w` or `+x`, you make (or keep) the file readable, writable or executable, respectively.
- With the options, `-r`, `-w` or `-x`, you make (or keep) the file nonreadable, nonwritable or nonexecutable, respectively.
- Depending on which system you work on, the previous options might change the permissions for everyone, or for just the owner of the file. To restrict the change to the owner of the file, to the members of the group to which the owner of the file belongs, and to the other users, prefix the option with `u` (like **u**ser), `g` (like **g**roup), or `o` (like **o**ther), respectively.
- The options can be combined. For instance, `chmod go-wx` *`file_name`* will prevent the members of the group and the other users to write and execute the file *`file_name`*.

2.2. **The tar command.** `tar` (for **t**ape **ar**chive) is used to put together a number of files into a single file, called an *archive*, possibly compressed so that it takes less space. It is also used to perform the inverse operation of creating a hierarchical structure of files from a single, possibly compressed, archive. Finally, it can be used to display the contents of an archive.

- You **c**reate a compressed (**z**ipped) archive
  - of all files stored in a directory `directory_name`, by executing
    `tar czf` *`archive_name`*`.tar.gz` *`directory_name`*
  - of the files *`filename_1`* ... *`filename_n`*, by executing
    `tar czf` *`archive_name`*`.tar.gz` *`filename_1`* ... *`filename_n`*

  which will create a **f**ile whose name is *`archive_name`*`.tar.gz`.
- You display the contents of an archive *`archive_name`*`.tar.gz` by executing `tar tzf` *`file_name`*`.tar.gz`, where `t` stands for **t**able of contents.
- You obtain the files from which an archive *`archive_name`*`.tar.gz` has been created by executing the command `tar xzf` *`archive_name`*`.tar.gz`, where `x` stands for e**x**tract.

Note the extensions we have been using: `.tar.gz` that indicates a compressed (`.gz`) archive (`.tar`). Sometimes, you will only want to compress or uncompress a single file; the commands `gzip` and `gunzip` will do the job, respectively.

3. Wildcards

Wildcards save you from typing too many characters. Here are some examples of uses of the wildcards `*`, `?` and `[`*`numbers_or_range_of_numbers`*`]`:

- `ls *` gives a listing of all files and directories in the working directory.
- `ls file*.py` gives a listing of all files whose name starts with `file` and ends in `.py`, with any characters (possibly none) in between (so it would match `file.py`, `file2.py`, ...).
- `ls file?3.py` gives a listing of all files whose name starts with `file` and ends in `3.py`, with exactly one character in between (so it would match `file13.py`, `fileA3.py`, ...).
- `ls file??.py` gives a listing of all files whose name starts with `file` and ends in `.py`, with exactly two characters in between (so it would match `file12.py`, `file1B.py`, ...).
- `ls file[13].py` gives a listing of all files whose name starts with `file` and ends in `.py`, with either `1` or `3` in between (so it would match `file1.py` and `file3.py`).
- `ls file[1-3].py` gives a listing of all files whose name starts with `file` and ends in `.py`, with either `1`, `2` or `3` in between (so it would match `file1.py`, `file2.py` and `file3.py`).

Of course, wildcard can be used with any command, not just `ls`.