Learning About UNIX-GNU/Linux

Module 1: Introduction

History

- UNIX was originally developed circa 1969 in AT&T Bell Labs. Key developers: Dennis Richie and Ken Thompson.
- Development was coupled to the invention of the C programming language, which allowed UNIX to be semi-portable to different hardware. (11,000 lines of portable C and 1000 lines of machine dependent assembler in those early days)
- As discussed below, UNIX includes a *kernel* and a number of small components and utilities built to work with the kernel.
- Circa 1974 the source was made available to selected Universities, including the U of T and especially Berkeley. This led to different "flavors" of UNIX. The code remained property of AT&T and the Universities signed non-disclosure agreements.
- In about 1979 various commericial vendors began to adopt UNIX under license from AT&T. The number of flavors increased (System V, BSD, HP-UX, Solaris, IRIX, etc.).
- In 1984 Richard Stallman drove the beginnings of the *Open Source* movement with the foundation of *GNU*. (GNU stands for "Gnu is Not Unix."). Later this became the *Free Software Foundation*. They began introduce open source products to work under UNIX.
    - One of their first and biggest successes was developing a C compiler, `gcc`, that was superior to any commercially available ones.

    - gcc probably still is the best C compiler in the world.
    - Virtually all of the proprietary utilities, shells, etc. that are associated with UNIX have now been re-written by *GNU* as Open Source.
- In 1991 21 year old Linus Torvalds wanted to buy a UNIX for his own computer but couldn't afford it. So he began writing a UNIX-like operating system called *Linux*. He made it Open Source.
    - With the *Linux* kernel and all of the GNU utilities available as Open Source, the *GNU/Linux* computing environment is at least as rich and powerful as the proprietary *UNIX* one.
        - Below we will occasionally refer to "UNIX/Linux," implying that UNIX and Linux are synonyms. From the standpoint of a user this is largely true. We should more properly refer to "UNIX-GNU/Linux" to give proper credit to GNU for their important role in the *Linux* computing environment.

Top

About "Open Source"

- There are various "Open Source" licenses.
    - The best known is the "General Public License" (GPL) from GNU. It is available at http://www.gnu.org/copyleft/gpl.html
    - Under the terms of the GPL, any person may obtain and change the code covered under the license, but must make those changes available at no charge to the world.
    - The terms of the license means, for example, that a commercial vendor of GNU/Linux, such as Red Hat, must make their distribution available at no charge.
        - Such vendors, then, remain commercially viable by making it convenient to buy their boxed distribution instead of downloading it from the web.
- Proponents of proprietary software, notably Microsoft, have attacked the Open Source movement as being:
    - Anarchistic
    - Anti-capitalism
    - Un-American
        - This one did not particularly impress people outside of the United States.
    - No good
- Advocates of the Open Source model point out that it is:
    - Anarchistic
    - Anti-monopoly
    - The collective effort of thousands of programmers around the world instead of a small development team inside a company
    - Good
- A collection pro-Open Source viewpoints may be found at: http://www.oreilly.com/catalog/opensources/book/toc.html.

Top

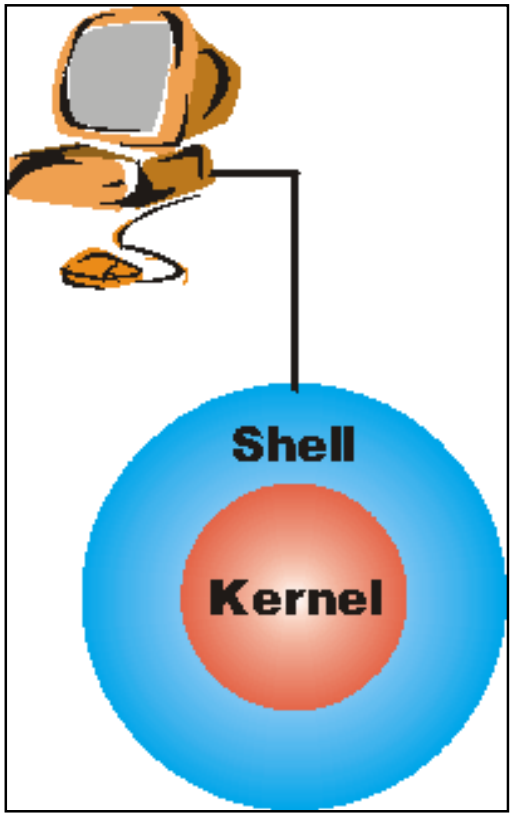
Philosophy of UNIX and GNU/Linux

- Multiuser - not patched in later
    - Every user needs an account
    - Need to *log in*.

- Multitasking - not patched in later.
- On top of the *kernel* are a number of small components.
    - Each component does one and only one thing.
    - Multiple components can be chained together.
- Everything is a file.
- The user is always right: don't bug them with lots of "Do you really really mean it?" confirmations.

Top

The Command Line Interface



- Windoze-like GUI's are exist, but we will not discuss them.
    - In the past year or so, both *Gnome* and *KDE* have become excellent desktops.
- The *shell* is the program that you interact with. It is between you and the UNIX/Linux *kernel*. It uses a *command line interface*. There are different shells, but we will use `bash`, which stands for "Bourne again shell."
    - One of the original shells was written by Steve Bourne of AT&T Bell Labs. It was named `sh`
- The shell maintains an idea of your *present working directory*. This distinguishes it from Windoze-like interfaces.
- *Everything* is customisable.
- The default *bash* prompt is:

```
[you@faraday pwd]$ _
```

  where **you** is your login name, **faraday** is the name of the computer, and **pwd** is name of your present working directory: we will discuss directory names further in Module 2. When you first log in, your present working directory is called your *home* directory, and typically has the same name as your login.
    - Throughout these documents, the underscore _ represents the cursor, waiting for you to type something.
- You enter a command by typing its name and pressing *Enter*. In the early days the input/output device was a teletype and many of the geeks were not good typists, so command names are often very short.
- Getting the date and time looks like:

```
[you@faraday pwd]$ date
Thu Apr 25 13:50:51 EDT 2002
[you@faraday pwd]$ _
```

  Note in the above example that the shell prompt is shown after the command has executed, so you may enter another command.

Top

Logging Out

- To log out, you just need to exit the shell. Type:

```
[you@faraday pwd]$ exit
```

- Alternatively, you can give the shell an "end of file" command, which tells it that the input is over. On a keyboard you send an end of file by simultaneously holding down the **Ctrl** key and the letter **d**. In documentation, the following forms are commonly used to indicate this sequence:
  - **`Ctrl+D`**: note that despite the notation it is a lower-case **d**.
  - **`^d`**
  - **`C-d`**

Top

Command Syntax

- Most commands take **arguments**
  - Some commands require them
  - For example, `echo` simply displays its arguments:

```
[you@faraday pwd]$ echo

[you@faraday pwd]$ echo hi there
hi there
[you@faraday pwd]$ _
```

- UNIX/Linux is case-sensitive, including commands:

```
[you@faraday pwd]$ echo whisper
whisper
[you@faraday pwd]$ ECHO SHOUT
bash: ECHO: command not found
[you@faraday pwd]$ _
```

- Many commands have **options**, which begin with a hyphen `-` or double hypens `--`. Options are given immediately after the name of the command and before any arguments. For example, `echo` has an option `-n` which means do not output the trailing newline.

```
[you@faraday pwd]$ echo -n hi there
hi there[you@faraday pwd]$ _
```

Top

Files and Directories

- Data can be stored in a file.
- There are directories where the files can be stored.
- Filename extensions such as the `.txt` in a file named `somefile.txt` are not enforced by UNIX/Linux. Some applications, written by Windoze wienies instead of UNIX geeks, require them however.
- Each file and directory has a **name**
  - A label used to refer to a particular file or directory
  - Permitted characters include letters, digits, hyphens (-), underscores (_), and periods (.)
    - For reasons to be explained later, the asterisk *****, although legal, should be avoided in filenames.
    - Support for spaces in filenames and directory names is limited and in general should be avoided. This is probably the *only* feature where Windoze is superior to UNIX/Linux.
  - Case-sensitive – `NewsCrew.mov` is different from `NewScrew.mov`
- Two commands can show you the names of the files and directories. The `lc` command sorts directories and files. For example, from my home directory the output is:

```
[harrison@faraday harrison]$ lc
Directories:
Desktop         INBOX           Mail            News            Office51
Storm           acct            bin             corres          courses
cv              db              etc             lab             lib
lpacct          mail            math            misc            ns_imap
nsmail          public_html     src             tmp             unixmenu
upscale

Files:
E-Mail_to_David_Harrison.doc  NetscapeAddressSave.ldif      abook.nab
[harrison@faraday harrison]$ _
```

The `ls` command mixes them all up:

```
[harrison@faraday harrison]$ ls
abook.nab  E-Mail_to_David_Harrison.doc  math                       src
acct       etc                           misc                       Storm
bin        INBOX                         NetscapeAddressSave.ldif   tmp
corres     lab                           News                       unixmenu
courses    lib                           ns_imap                    upscale
cv         lpacct                        nsmail
db         mail                          Office51
Desktop    Mail                          public_html
[harrison@faraday harrison]$ _
```

  - Depending on how your account has been set up, the output from the `ls` command may include colors to indicate the type of file.
  - You may wish to know that `lc` is not standard with UNIX/Linux, but many many people, including me, carry it around with them to whatever machine they are working on.

Top

Creating Files - Editors

- Typically to create a new file one uses an editor. The choices include:
  - **`pico`**: a very simple editor. To use you will need to remember that we are using a notation that **`^X`**, for example, means to simultaneously hold down the **Ctrl** key while pressing the letter **x** This notation is used by the editor to remind you of how to use it.
  - **`vi`**: a very powerful editor with a very steep learning curve. This is my editor of choice.
  - **`emacs`**: a very very powerful editor with a steep learning curve. If I were starting over this is the editor I would use. For simple use, it is easier to learn than `vi`.
- We shall create a file named `shopping_list`, and instead of using an editor shall use the `cat` command. In the exercises you may choose to instead use this as an opportunity to learn an editor.

```
[you@faraday pwd]$ cat > shopping_list
ham
spam
kolbassa
^d
[you@faraday pwd]$ _
```

  You may wish to know that the command `cat` is short for *concatenate*, which means to link together.
  - One of the more complex pieces of software in UNIX/Linux is a *device driver*. An old joke is that a true guru can write a device driver with `cat`.
- Note the greater-than sign (>) – this is necessary to create the file
- The text typed is written to a file with the specified name
- Press `Ctrl+D` after a line-break to denote the end of the file
  - The next shell prompt is displayed
- `ls` or `lc` demonstrates the existence of the new file

Top

Displaying File Contents

- In UNIX/Linux there are many ways to do almost every task. Similarly, there are many ways to display the contents of a file.
- The `cat` command will display the contents of a file whose name is given as an argument:

```
[you@faraday pwd]$ cat shopping_list
ham
spam
kolbassa
[you@faraday pwd]$ _
```

- Since filename extensions do not necessarily indicate the type of contents that are in a file, using `cat` on, say, a binary file can give ghastly results. The `file` command looks inside its argument to figure out what kind of contents are in it:

```
[you@faraday pwd]$ file shopping_list
shopping_list: ASCII text
[you@faraday pwd]$ _
```

- If the file were very long using `cat` would cause the contents to scroll off the top of your screen, only showing you the last 24 lines or so. Two standard "pagers" that show only a screenful at a time are named `more` and `less`; we shall discuss these and how to learn about them later.

Top

Appending to a File

- From the shell, double greater-than symbols (>>) append to a file:

```
[you@faraday pwd]$ cat >> shopping_list
broccoli
beer
^d
[you@faraday pwd]$ cat shopping_list
ham
spam
kolbassa
broccoli
beer
[you@faraday pwd]$ _
```

Top

Deleting Files With `rm`

- To delete a file, use the `rm` ('remove') command
- Simply pass the name of the file to be deleted as an argument:

```
[you@faraday pwd]$ rm shopping_list
[you@faraday pwd]$ _
```

- The file and its contents are removed
  - There is no recycle bin
  - There is no 'unrm' command

Top

UNIX Command Feedback

- Typically, successful commands do not give any output

- Messages are displayed in the case of errors
- The `rm` is typical
  - If it manages to delete the specified file, it does so silently
  - There is no 'File shopping list has been removed' message
  - But if the command fails for whatever reason, a message is displayed
- The silence can be be off-putting for beginners
- It is standard behavior, and doesn't take long to get used to

Changing Your Password

- A good password in one that:
  - You will never ever forget.
  - Your best friend will not be able to guess.
  - Contains at least one character that is not a letter of the alphabet.
- Some sites have *password aging* so that you must change your password periodically.
- The password command is `passwd`

```
[you@faraday pwd]$ passwd _
```

  - After you press `Enter` you are prompted for your current password:

```
[you@faraday pwd]$ passwd
Changing password for you
(current) UNIX password: _
```

    - This is a desirable security precaution.
    - Anything you type will not be shown on the screen.
  - After successfully giving your current password you will be prompted for the new one:

```
[you@faraday pwd]$ passwd
Changing password for you
(current) UNIX password:
New UNIX password: _
```

    - Whatever you type will not be shown.
    - After enterring the new password, you will be prompted to enter it again.
      - This is in case you mis-typed your new password the first time.

Copying and Renaming Files with `cp` and `mv`

- To copy the contents of a file into another file, use the `cp` command:

```
[you@faraday pwd]$ cp CV.pdf old-CV.pdf
[you@faraday pwd]$ _
```

- To rename a file use the `mv` ('move') command:

```
[you@faraday pwd]$ mv committee minutes.txt
[you@faraday pwd]$ _
```

- For both commands, the existing name is specified as the first argument and the new name as the second
  - If a file with the new name already exists, it is overwritten

Filename and Command Name Completion

- The shell can making typing filenames and command names easier
- Once an unambiguous prefix has been typed, pressing `Tab` will automatically 'type' the rest
- For example, after typing this:

```
[you@faraday pwd]$ rm sho_
```

Pressing `Tab` may turn it into this:

```
[you@faraday pwd]$ rm shopping_list_
```

- This also works with command names. Typing:

```
[you@faraday pwd]$ ec_
```

and pressing `Tab` turns it into:

```
[you@faraday pwd]$ echo_
```

  - Pressing the `Tab` ley twice in succession will give you a list of all the completions of the filename or command name.

Command History

- Often it is desired to repeat a previously-executed command
- The shell keeps a **command history** for this purpose
  - Use the Up and Down arrow keys to scroll through the list of previous commands
  - Press `Enter` to execute the displayed command
- Commands can also be edited before being run
  - Particularly useful for fixing a typo in the previous command
  - The Left and Right arrow keys navigate across a command
  - Extra characters can be typed at any point
  - `Backspace` deletes characters to the left of the cursor

- `Del` and `Ctrl+D` delete the character under the cursor
  - If your cursor is a vertical line instead of a block, `Del` and `Ctrl+D` delete the character to its right.
- Take care not to log out by holding down `Ctrl+D` too long

Exercise 1

- Log in
- Use the `lc` or `ls` command to see if you have any files
- Log out
- Log in again
- Create a new file by doing one or more of the following:
  - Create a file with `cat`:

```
[you@faraday pwd]$ cat > hello.txt
Hello world!
This is a text file.
^d
[you@faraday pwd]$ _
```

Press `Enter` at the end of the last line, then `Ctrl+D` (**^d**) to denote the end of the file.

  - Create a file with `pico`:

```
[you@faraday pwd]$ pico hello.txt
```

Use the editor to create the text:

```
Hello world!
This is a text file.
```

    - Note there is no greater-than sign (>) in the command.
  - Get hold of a reference on the editor `vi`. Then:

```
[you@faraday pwd]$ vi hello.txt
```

Use the editor to create the text:

```
Hello world!
This is a text file.
```

  - Get hold of a reference on the editor `emacs`. Then:

```
[you@faraday pwd]$ emacs hello.txt
```

Use the editor to create the text:

```
Hello world!
This is a text file.
```

- Use `ls` or `lc` to see that the file exists.
- Display the contents of the file.

- Display the file again, but use the cursor keys to execute the same command again without having to retype it.

Exercise 2

- Create a second file. Call it `secret-of-the-universe`, and put in it whatever content you deem appropriate.
- Check that the contents of the file is ASCII text using the `file` command.
- Display the contents of this file. Minimise the typing needed to do this:
  - Scroll back through the command history to the command you used to create the file.
  - Change that command to display `secret-of-the-universe` instead of creating it.

Exercise 3

- Copy `secret-of-the-universe` to a new file called `answer.txt`. Use `Tab` to avoid typing the existing file's name in full.
- Now copy `hello.txt` to `answer.txt`. What's happened now?
- Delete the original file, `hello.txt`.
- Rename `answer.txt` to `message`.
- Try asking `rm` to delete a file called `missing`. What happens?
- Try copying `secret-of-the-universe` again, but don't specify a filename to which to copy. What happens now?