Learning About UNIX-GNU/Linux

Module 2: Getting Started

About Shells

- The are many shells. Some common ones, in historical order, are:
  - `sh`: the original Bourne shell.
    - Written by Steve Bourne at AT&T Bell Labs.
    - Provides a shell programming language.
  - `csh`: the "C shell."
    - Written by Bill Joy et al. at the University of California, Berkeley.
    - Added some features to improve the user interface.
    - Developed another shell programming language, similar to C.
    - Many bugs in that programming language.
  - `tcsh`: Tenex C shell.
    - Tenex was an operating system championed by Digital Equipment Corporation.
    - C shell like syntax with many bug fixes and Tenex-like extensions.
  - `ksh`: the "Korn shell."
    - Written by David Korn at AT&T Bell Labs.
    - An extension of the Bourne shell.
    - Added some of the improved user-interface of the C shell.
    - Proprietary.
  - `bash`: Bourne again shell.
    - Developed as Open Source by GNU.

        - All the features of `sh` and `ksh` plus more.
- On many systems the file `/etc/shells` lists the available shells.
- `bash` and `tcsh` are the most popular shells for user-interaction.
- `sh-ksh-bash` syntax is the most popular for programming.
- The program `chsh` allows you to change your login shell.
- In this module, most examples will work with any of the well-known shells.
- In Module 3 we will be discussing using `sh-ksh` and especially `bash`.
- For further information see http://www.faqs.org/faqs/unix-faq/shell/shell-differences/.

Top

Files and Directories

- A **directory** is a collection of files and/or other directories
    - Because a directory can contain other directories, we get a directory **hierarchy**
- The ‘top level’ of the hierarchy is the **root directory**
- Files and directories can be named by a **path**
    - The root directory is referred to as /
    - Other directories are referred to by name, and their names are separated by /
        - Why MicroSoft later chose to use a backslash \ instead of the already established / as a directory separator is a mystery.
- If a path refers to a directory it can end in /

Top

Examples of Absolute Paths

- An **absolute path** starts at the root of the directory hierarchy, and names directories under it
- In the root directory is a directory called bin, which contains a file called ls:

  ```
  /bin/ls
  ```

- The following example will run the ls command, by specifying the absolute path to it:

  ```
  [you@faraday you]$ /bin/ls
  ```

    - If you give an argument to `ls` it can be either a directory name or a filename
- The `lc` command is in the directory `/usr/local/bin/` so to run it with an absolute path you type:

  ```
  [you@faraday you]$ /usr/local/bin/lc
  ```

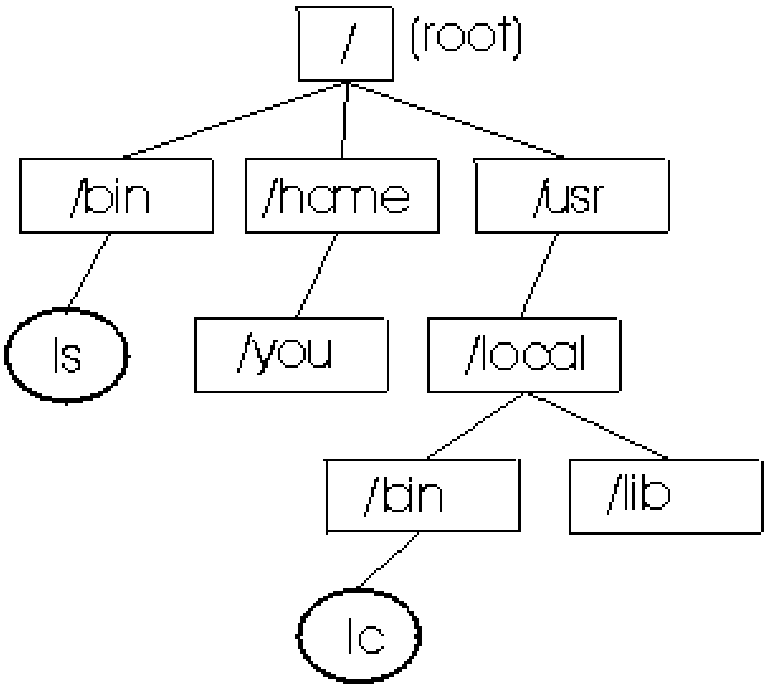
- We can use `lc` to list files in a specific directory by specifying the absolute path:

  ```
  [you@faraday you]$ lc /usr/local/lib
  ```

    - If you give an argument to `lc` it must be a directory name
    - We will discuss in Module 3 how the shell found the `lc` command in the previous example.

- On many UNIX/Linux systems, the user's *home* directory is `/home/you/` where `you` is typically your login name.

The figure to the right indicates some of the directories and files that may be found on a typical UNIX/Linux implementation. Entries in rectangles are directories, entries in ellipses are files:



Top

Present Working Directory

- As mentioned in the previous Module, the shell maintains an idea of your present working directory – the directory in which you are working.
- By default, the *bash* prompt displays the name of the directory that is your present working directory as the last item.
- Commands like `ls` use the current directory if none is specified
- Use the pwd command to see what your current directory is:

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

- Change the current directory with `cd`:

```
[you@faraday you]$ cd /usr/local
[you@faraday local]$ pwd
/usr/local
[you@faraday local]$ _
```

- Use `cd` with no argument to get back to your home directory

```
[you@faraday local]$ cd
[you@faraday you]$ _
```

- Use `cd -` to return to your previous working directory.

```
[you@faraday you]$ cd /usr/local
[you@faraday local]$ cd /tmp
[you@faraday tmp]$ cd -
[you@faraday local]$ pwd
/usr/local
[you@faraday local]$ _
```

Top

Making and Deleting Directories

- The `mkdir` command makes new, empty, directories
- You may only make directories where you have *write permission* for the present working directory. We will discuss permissions more later.
- To make a directory for storing company accounts:

```
[you@faraday you]$ mkdir Accounts
[you@faraday you]$ _
```

- To delete an empty directory, use `rmdir`:

```
[you@faraday you]$ rmdir Accounts
[you@faraday you]$ _
```

- To delete a directory that is not empty you may change into the directory and remove all its contents, change back to the *parent* of the directory, and then use `rmdir`
- Another way to delete a non-empty directory is to the use `rm -r`

```
[you@faraday you]$ rm -r NonEmptyDirecotory
[you@faraday you]$ _
```

  - The `r` means "recursive."
  - It removes *all* the files and directories in the named argument.
- Be careful: `rm` can be a dangerous tool if misused. `rm -r` is even more dangerous.

Top

Relative Paths

- Paths don't have to start from the root directory
  - A path which doesn't start with / is a **relative path**
  - It is relative to some other directory, usually the current directory
- For example, the following sets of directory changes both end up in the same directory:

```
[you@faraday you]$ cd /usr/share/doc
[you@faraday doc]$ _
```

```
[you@faraday you]$ cd /
[you@faraday /]$ cd usr
[you@faraday usr]$ cd share
[you@faraday share]$ cd doc
[you@faraday doc]$ _
```

- Relative paths specify files inside directories in the same way as absolute ones. The following two sets of commands both execute the program `lc` in the directory `/usr/local/bin`

```
[you@faraday you]$ /usr/local/bin/lc
```

```
[you@faraday you]$ cd /usr/local/
[you@faraday local]$ bin/lc
```

Top

Special Dot Files and Directories

- Every directory contains two special subdirectories which help making relative paths
- The directory `..` points to the parent directory, so to list the files in the directory which contains the current directory, use `ls ..`
  - For example, if we start from the directory `/home/you`

```
[you@faraday you]$ pwd
/home/you
[you@faraday you]$ cd ..
[you@faraday home]$ pwd
/home
```

- The special directory `.` points to the present working directory.
  - So `./foo` is the same file as `foo`
- There can also be files and directories whose name begins with a period, such as `.profile`
- These "dot files" and "dot directories" are *hidden* in the sense that neither `ls` or `lc` show them by default.
  - Nothing else about these files and directories are special.
  - They are often used for configuration.
- You can see the existence of the dot files and directories, along with all the other files and directories, by giving options to `ls` and `lc`:
  - `ls -a` shows all files and directories
  - `lc -h` shows the hidden files and directories along with all the others

Top

Paths to Home Directories

- The tilde symbol ~ is an abbreviation for your home directory
  - For user `you` the following are equivalent:

```
[you@faraday some_directory]$ cd /home/you/documents
[you@faraday documents]$ _
```

```
[you@faraday some_directory]$ cd ~/documents
[you@faraday documents]$ _
```

- You can get the paths to other users' home directories using ˜. This for a user `alice`:

```
[you@faraday some_directory]$ cat ~alice/somefile
```

- The following are all the same for user `you`

```
[you@faraday some_directory]$ cd
```

```
[you@faraday some_directory]$ cd /home/you
```

```
[you@faraday some_directory]$ cd ~
```

Top

Specifying Multiple Files

- Many programs can be given a list of files
  - To delete several files at once:

```
[you@faraday some_directory]$ rm oldnotes.txt tmp.txt
[you@faraday some_directory]$ _
```

  - To make several directories in one go:

```
[you@faraday some_directory]$ mkdir Accounts Reports
[you@faraday some_directory]$ _
```

- The original use of `cat` was to join multiple files together
  - For example, to list two files, one after another:

```
[you@faraday some_directory]$ cat file1 file2
```

- If a filename contains spaces, or characters which are interpreted by the shell (e.g., *, ˜), put single quotes around them:

```
[you@faraday some_directory]$ rm 'Strawberry Fields.mp3'
[you@faraday some_directory]$ cat '* important notes.txt *'
```

Top

Specifying Files with Wildcards

- Use the * wildcard to specify multiple filenames to a program:

```
[you@faraday some_directory]$ ls *.txt
foo.txt bar.txt
[you@faraday some_directory]$ _
```

- The shell expands the wildcard, and passes the full list of files to the program, in the above example to `ls`
- Just using * on its own will expand to all the files and directories in the current directory. Thus the following command removes all the files in the present working directory!

```
[you@faraday some_directory]$ rm *
[you@faraday some_directory]$ _
```

  The following, then, will remove all files and sub-directories of the present working directory!

```
[you@faraday some_directory]$ rm -r *
[you@faraday some_directory]$ _
```

  - When the shell expands file and directory names with `*`, it does not include names that begin with a period `(.)`. A moment's reflection will convince you that this is a *good thing*.

Top

Looking at Files

- You can use an editor to look at the contents of an ASCII text file.
- You can use `cat` to output a small ASCII file to your screen. A long file, such as Faraday's password file `/etc/passwd`, will scroll off the top of your terminal window leaving only the last 24 lines or so visible.
- From an X-terminal, you may have *scroll bars* on the terminal window so you may see what has scrolled off the top of the screen. They may be moved using the middle button of the mouse.
  - For a two-button mouse, holding down both buttons simultaneously simulates the middle button.
- You can look at the first few lines of a file with `head`

```
[you@faraday some_directory]$ head /etc/passwd
```

- You may see the last few lines of a file with `tail`

```
[you@faraday some_directory]$ tail /etc/passwd
```

  - The `-f` option to `tail` follows the file as it grows. So to watch accesses to Faraday's web server, you can follow the log file `/usr/local/apache/logs/access_log`:

```
[you@faraday some_directory]$ tail -f
/usr/local/apache/logs/access_log
```

  - Send an *interrupt* to `tail` to stop the program with `Ctrl-C`

- You may use the pagers `more` or `less` to scroll through a file a screenful at a time.

```
[you@faraday some_directory]$ more /etc/passwd
```

```
[you@faraday some_directory]$ less /etc/passwd
```

  - To quit the output press `q`
  - `more` was written first; `less` has similar functionality with more power.
  - Both programs have help available by pressing `h`
- You can view lines in a file that contain a particular string with `grep`

```
[you@faraday some_directory]$ grep 'harrison' /etc/passwd
```

  - The origins of the name `grep` are the original UNIX editor `ed`. It stands for "global regular expression print." We will discuss regular expressions in more detail later.
- For a binary non-text file, the program `od` will show its contents in octal or hexidecimal
- For a binary non-text file, the program `strings` extracts and displays the ASCII strings that are in the file.

Top

Finding Documentation for Programs

- Use the `man` command to read the manual for a program
- The manual for a program is called its **man page**
- To read a man page, specify the name of the program as an argument to `man`:

```
[you@faraday some_directory]$ man mkdir
```

- To quit from the man page viewer press `q`
  - `man` pipes its output to `less`.
    - The next section discusses the meaning of the word "pipes" in the previous sentence.
- Man pages for programs usually have the following information:
  - A description of what it does
  - A list of options which it accepts
  - Other information, such as the name of the author
- The man pages are divided into sections. For example, Section 1 is for user commands, Section 5 is for file formats, Section 8 is for administration commands. If the `man` program is given two arguments, the first is the section and the second is the name of the man page. It searches the sections in numerical order.
  - To get the man page for the `passwd` command that is used to change your password the following are equivalent:

```
[you@faraday some_directory]$ man passwd
```

```
[you@faraday some_directory]$ man 1 passwd
```

  - To get the man page that describes the format of the password file itself use:

```
[you@faraday some_directory]$ man 5 passwd
```

- The system maintains a database of keywords in the man pages. You can search for man pages for a given keyword with:

```
[you@faraday some_directory]$ man -k keyword
```

- A synonym for `man -k` is the command `apropos`:

  ```
  [you@faraday some_directory]$ apropos keyword
  ```

- There is lots of documentation available on-line. Most people automatically pipe the output to a pager:

  ```
  [you@faraday some_directory]$ apropos keyword | more
  ```

  - This "pipe" operation using the vertical bar **|** is the topic of the next section.

- GNU has written an alternative to the `man` command called `info`:

```
[you@faraday some_directory]$ info mkdir
```

  - Although the `man` command exists on virtually every UNIX/Linux flavor, `info` usually appears only with relatively modern Linux distributions.
    - On modern Linux distributions there are beginning to appear GNU programs which do not have man pages but only info ones.
    - With such distributions there is almost always an `info` page for all programs which have a man page.
  - The `info` system includes nodes, menus, etc.
    - The top of screen always shows the current node, the next and previous nodes if any, and the parent "up" node if any. Simple navigation of nodes can be done with:

Keystroke	Action
space	Scroll to next screenful like `more` and `less`
`n`	Go to next node
`p`	Previous node
`u`	Up to parent node
`b`	Back to the beginning of the current node
`q`	Quit `info`
`h`	Invoke an on-line tutorial on `info`

      - If there is a next node after the current one, pressing the space bar at the end of a node will take you to the next node. Otherwise it starts over from the beginning of the current node.
  - For simple cases, such as looking at the info page for `mkdir`, `info` is almost identical to the `man` command.
  - Menus are identified by an asterisk `*`:

```
Here is some text in a made-up info page.

The line "* Menu:" below indicates the beginning
of a menu,  and the lines that come after it are
the menu items. The items are identified by
beginning with an asterisk *.

In this example those items are named
"Choice 1" and "Choice 2"

* Menu:

* Choice 1::      Description of this menu choice.
* Choice 2::      Description of this second choice.

This line is not part of the menu.
```

- There are two ways to access a particular menu item:
  - Use the arrow keys on the keyboard to scroll to the desired choice and press the `Enter` key.
  - Press the `m` key. A line will open at the bottom of the screen in which you can type the name of the desired item.
    - You can complete partially typed names by pressing `Tab`. If the completion is not unique pressing `Tab` a second time shows the choices.
    - This is identical to the completion syntax for the shell that we learned about in Module 1.

Top

Plumbing

- As already mentioned, UNIX/Linux has many many small utilities
  - Each utility is designed to do one and only one thing
- The output of a utility is directed to a stream called `stdout`, for "standard output."
- For many utilities, if not given an argument it reads a stream called `stdin` (for "standard input") for its input
  - When we created files using `cat` in Module 1, we were reading the keyboard to supply the characters via `stdin`, which `cat` then directed to the file.
- The utilities can be combined together
  - `cat` outputs the file that is its argument.
  - `wc` counts the lines, words and characters in a file.

```
[you@faraday some_directory]$ wc /etc/passwd
2786 3199 161149
[you@faraday some_directory]$ _
```

  - We can "pipe" the output of `cat` to `wc` with the pipe symbol **|**

```
[you@faraday some_directory]$ cat /etc/passwd | wc
2786 3199 161149
[you@faraday some_directory]$ _
```

    - Since `wc` was not given an argument, it is reading `stdin`, which is the `stdout` of the `cat` command.

- The pieces of a pipeline do not have to be on a single line.
    - Enter the first part of the command, being sure to end in the pipe symbol:

      ```
      [you@faraday some_directory]$ cat /etc/passwd | _
      ```

        - This will not work with `csh` or `tcsh`.
    - When your press `Enter` the shells *secondary prompt* is displayed:

      ```
      [you@faraday some_directory]$ cat /etc/passwd |
      > _
      ```

    - You can then enter the next command and press `Enter`:

      ```
      [you@faraday some_directory]$ cat /etc/passwd |
      > wc
      2786 3199 161149
      [you@faraday some_directory]$ _
      ```

- Virtually all UNIX/Linux commands can be similarly combined.
    - `who` lists the users currently logged in.
    - `wc -l` only outputs the number of lines in its input.
    - We can count the number of users currently logged in with:

      ```
      [you@faraday some_directory]$ who | wc -l
      11
      [you@faraday some_directory]$ _
      ```

- `stdout` can be directed to a file with the greater-than symbol >

  ```
  [you@faraday some_directory]$ who | wc -l > somefile
  [you@faraday some_directory]$ cat somefile
  11
  [you@faraday some_directory]$ _
  ```

    - This syntax was also used to create a file using `cat` in Module 1.
- The shell can direct `stdin` from a file with the less-than symbol <

  ```
  [you@faraday some_directory]$ wc -l < /etc/passwd
  2786
  [you@faraday some_directory]$ _
  ```

Top

More About Command Options

- Some commands have multiple options:
    - `who` accepts a `-i` flag which adds the idle time for each user

      ```
      [you@faraday some_directory]$ who -i
      ```

    - It also accepts a `-H` flag which adds a label to each column of its output:

```
[you@faraday some_directory]$ who -H
```

- The options can be combined. All of the versions below are equivalent:

```
[you@faraday some_directory]$ who -i -H
```

```
[you@faraday some_directory]$ who -H -i
```

```
[you@faraday some_directory]$ who -iH
```

```
[you@faraday some_directory]$ who -Hi
```

- Some command options require further arguments.
  - The `last` command lists all logins since the logs were initialised. Usually the following command lists many many logins:

```
[you@faraday some_directory]$ last
```

  - A `-n num` flag only lists the last `num` logins:

```
[you@faraday some_directory]$ last -n 10
```

    - For well-behaved programs, the space between the option and its argument is optional:

```
[you@faraday some_directory]$ last -n10
```

    - The effect of the above command is almost identical to:

```
[you@faraday some_directory]$ last | head
```

    - In the early days of UNIX, the space between the option and its argument was not allowed. Sadly, some of this ancient code survives today.
  - The `-R` flag suppresses the listing of the machine the user logged in from:

```
[you@faraday some_directory]$ last -R
```

- Command line options with and without arguments can be combined. All the following forms are identical:

```
[you@faraday some_directory]$ last -R -n10
```

```
[you@faraday some_directory]$ last -Rn 10
```

```
[you@faraday some_directory]$ last -Rn10
```

```
[you@faraday some_directory]$ last -n 10 -R
```

- In the early days of UNIX, command options were a single letter preceded by a single hyphen. Now options can be multiple letter strings preceded by two hyphens.
  - The `who` command accepts a long option `--version` which prints its version number.

```
[you@faraday some_directory]$ who --version
```

  - The `date` command accepts a `-d` flag for manipulating the date shown. The following command shows the date 2 days ago:

```
[you@faraday some_directory]$ date -d '2 days ago'
```

- There is a long synonym of the `-d` option named `--date`.

  ```
  [you@faraday some_directory]$ date --date='2 days ago'
  ```

  - Note that in the long version an equal sign = connects the option with its argument.

- The man pages for options follow several different conventions.
  - In older man pages which document only options that begin with a single hyphen, a fragment of the man page can look like this:

    ```
    SYNOPSIS
        last [-R] … [-n num] [-adiox] …
    …
    OPTIONS
        -n num  Show num lines
        -R      Suppresses the display of the hostname field
    …
        -x      Display  the  system shutdown entries and run level
                changes.
    ```

    - The ellipsis … indicate parts of the man page not duplicated above.
    - Entires surrounded by square brackets, `[ … ]` are options.
    - Commands that require arguments will list them not surrounded by square brackets.
    - The `last` command also accepts the options `-a`, `-d`, `-i`, and `-o`. The full man page document these too.
    - Some programs have on-line help which looks similar to the synopsis.
  - A common format for GNU programs looks like this:

    ```
    SYNOPSIS
        cut [OPTION]... [FILE]...

    DESCRIPTION
        Print  selected  parts of lines from each FILE to standard
        output.
    …
        -d, --delimiter=DELIM
                use DELIM instead of TAB for field delimiter
        -f, --fields=LIST
                output only these fields;  also print any line that
                contains  no  delimiter  character,  unless  the -s
                option is specified
    ```

    - Since the long form options `--delimiter` and `--fields` require arguments, it is understood that the short form equivalents do also. In the old form these would have been described as:

      ```
          -d DELIM   Use DELIM instead of TAB for field delimiter
          -f LIST    Output only these fields;  also print any
                     line that contains  no  delimiter character,
                     unless  the -s option is specified
      ```

Top

Exercise 1

- From your home directory execute `pwd` and note the name of the *parent* of your home directory. Below we shall call that directory `/home` but your system may be configured differently.
- Use `cd` to go to the root directory. View its contents.
- Change to the `/home` directory and view its contents. Typically these are the home directories of all users except for the master user *root*.
  - On Faraday, the student users' home directories are located elsewhere.
- Use `grep` on /etc/passwd to extract the line corresponding to one of the directories of `/home`. See if you can find the part of the line that specifies that user's home directory.
- Use the man page on the format of the password file to determine the meaning of the other fields in the line that you extracted.
- Change to your home directory and confirm that that is where you are.
- Use the `hostname` command, with no options, to print the hostname of the machine you are using.
- Use `man` to display some documentation on the `hostname` command. Find out how to make it print the aliases for the machine. You will need to scroll down the manpage to the ‘Options’ section.

Top

Exercise 2

- Create a text file in your home directory called `shakespear`, containing the following text:

```
Shall I compare thee to a summer’s day?
Thou art more lovely and more temperate
```

- Rename it to `sonnet-18.txt`.
- Make a new directory in your home directory, called `poetry`.
- Move the poem file into the new directory.
- The * wildcard on its own is expanded by the shell to a list of all the files in the current directory. Use the echo command to see the result (but make sure you are in a directory with a few files or directories first)
- Use quoting to make echo print out an actual * symbol.
- In the `poetry` directory create another file, `sonnet-29.txt`:

```
When in disgrace with Fortune and men’s eyes,
I all alone beweep my outcast state,
```

- Use the `cat` command to display both of the poems, using a wildcard.
- Create a directory in your home directory named `tmp`
- Copy all the files in the `poetry` directory into the `tmp` one.
- Finally, use the `rm` command to delete the poetry directory and the poems in it.

Top

Exercise 3

- Use `grep` on `/etc/passwd` to see your entry. Identify your login shell.
- Combine `grep` and `wc` to count the number of users who use the same shell as you for their login shell.
- Learn about the `-v` flag to `grep` from its man page. Use it to count the number of users who do not use your login shell.
- Use the utility `sort` to list the users of your login shell, sorted by their login.
- Use `cut` to display a sorted list of these users that displays only their login.
    - Now you can begin to appreciate the power of having a command history available to avoid typing the same things over and over!
    - Hint: as the man page says, by default `cut` uses Tabs as the field separators. This is not what is used in `/etc/passwd`. Thus you will need a `-d` flag to set the delimiter. You will also need a `-f` flag to tell it which field to cut out and send on to `stdout`.

Top

This is $Revision: 1.16 $, $Date: 2002/06/20 20:17:25 $ (year/month/day UTC).