THE UNIX COMMAND-LINE AND C0

ANAND SUBRAMANIAN
<asubrama@andrew.cmu.edu>

May 21, 2012
Introduction

- You will be compiling and running code on Andrew Linux.
- Need familiarity with a *nix command line interface.
- Get one by opening a terminal.
- Alternately, get one via a secure shell connection (later).
CLI vs GUI

- Consists of a shell which accepts textual input from the user.
- Shell is a Read-Evaluate Loop.
- Predates GUIs.
- Easier to design and automate.
- We will be testing your programs using the command line.
- Still widely preferred for logging in via network.
Available shells

- Input can be a program written in the shell’s programming language.
- Many different shells. Many different languages.
Available shells

- Input can be a program written in the shell’s programming language.
- Many different shells. Many different languages.
- All execute commands like:
  
  \[
  \text{command [arg1] \ldots [arg n]}
  \]

- For example, this says what shell you use:
  
  \[
  > \text{ps -p $$}
  \]
Available shells

- Input can be a program written in the shell’s programming language.
- Many different shells. Many different languages.
- All execute commands like:
  `command [arg1] ... [arg n]`
- For example, this says what shell you use:
  `> ps -p $$`
- We can use `bash` or `csh`.
- Use `chsh` to change shell.
Context: *nix File System

- *nix file system is part of the shell’s context.
- Hierarchical filesystem.
Dirs or Files identified by absolute path or relative path.

Absolute path begins with `/`.

Shell has a **Current Working Directory**:

```bash
> pwd
```

Relative path is offset from `pwd`. 
Paths

- Dirs or Files identified by absolute path or relative path.
- Absolute path begins with `/`.
- Shell has a **Current Working Directory**:
  > `pwd`
- Relative path is offset from `pwd`.
- `..` refers to the directory containing the current working directory.
- `~` is short-hand for your home directory’s absolute path.
- `~username` can be used for home-dir of any user.
- `.` is short hand for current working directory’s absolute path.
Practice: `cd`, `ls`, etc..

- `cd`: change working directory
- `ls`: list contents of directory
- `mkdir`: make directory
- `touch`: “touch” a file
- `cp`: copy file or directory
- `mv`: move/rename file or directory
- `rm`: remove file or directory (caution)
- `echo`: print to stdout
- `cat`: dump file to screen
- `less`: view part of file
Getting help: Dog’s best friend

- Many commands accept `-h` or `--help` as an argument.
- Manual Pages:
  `man command`
- Example:
  
  ```
  > man ls
  ```
- Info pages: some commands have these. Relatively uncommon.
- Practice navigating a few man pages.
which command?

- Some commands are shell intrinsics. Most are executables in the filesystem.
- Shell searches the paths stored in an Environment Variable called PATH.
- Alternately, name executables using absolute path. Example:
  > /bin/ls
  > ./bin_in_my_current_working_dir
- Which executable are you using?
  > which cd
  > which which
- Not very nice if the path gets corrupted, is it?
The Shell Config File

- Need to add 15-122 commands to executable paths.
- Need to edit shell config file.
Need to add 15-122 commands to executable paths.

Need to edit shell config file.

Need a text editor! (Try emacs or vi)

Open your shell’s config file:
  > emacs ~/.bashrc
  > emacs ~/.cshrc
The PATH variable

- Add the following to your config file:
  
  setenv PATH
  
  `$\{PATH\}:/afs/andrew/course/15/122/bin/`  #csh
  
  export
  
  `PATH=$\{PATH\}:/afs/andrew/course/15/122/bin/`  #bash

  
  Reload config:
  
  `csh> source ~/.cshrc`
  
  `bash> source ~/.bashrc`

  
  Confirm with:
  
  `csh> env`
  
  `bash> echo $\{PATH\}`
The PATH variable

- Add the following to your config file:
  ```bash
  setenv PATH
  ${PATH}:/afs/andrew/course/15/122/bin/  #csh
  export
  PATH=${PATH}:/afs/andrew/course/15/122/bin/  #bash
  ```
- Reload config:
  ```bash
  csh> source ~/.cshrc
  bash> source ~/.bashrc
  ```
The PATH variable

- Add the following to your config file:
  ```
  setenv PATH
  ${PATH}:/afs/andrew/course/15/122/bin/  #csh
  export
  PATH=${PATH}:/afs/andrew/course/15/122/bin/  #bash
  ```
- Reload config:
  ```
  csh> source ~/.cshrc
  bash> source ~/.bashrc
  ```
- Confirm with:
  ```
  csh> env
  bash> echo $PATH
  ```
Unix Permissions

- Unix security model has **users** which belong to **groups**.
- Each file has a distinguished **owning user** and **owning group**.
- Additionally, each file has permission bits.
- Useful commands: `chmod`, `chown`, `chgrp`. 

<table>
<thead>
<tr>
<th>Read</th>
<th>Execute</th>
<th>Write</th>
<th>SUID</th>
<th>SGID</th>
<th>Sticky</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
Unix Permissions

- Unix security model has **users** which belong to **groups**.
- Each file has a distinguished **owning user** and **owning group**.
- Additionally, each file has permission bits.
- Useful commands: `chmod`, `chown`, `chgrp`.
- Example of permissions matrix:

<table>
<thead>
<tr>
<th></th>
<th>Read</th>
<th>Execute</th>
<th>Write</th>
<th>SUID</th>
<th>SGID</th>
<th>Sticky</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User</strong></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>Group</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
AFS Permissions

- AFS maintains separate permissions for each user and each directory.
- `fs la` and `fs sa` are your friends.
AFS Permissions

- AFS maintains separate permissions for each user and each directory.
- `fs la` and `fs sa` are your friends.
- Exercise: how do you find more info about the `fs` command?
AFS Permissions

▶ AFS maintains separate permissions for each user and each directory.
▶ `fs la` and `fs sa` are your friends.
▶ Exercise: how do you find more info about the `fs` command?
▶ Examples:
  > `fs la ~/public`
  > `fs la ~/private`
AFS Permissions

- AFS maintains separate permissions for each user and each directory.
- `fs la` and `fs sa` are your friends.
- Exercise: how do you find more info about the `fs` command?
- Examples:
  - `> fs la ~/public`
  - `> fs la ~/private`
- All work is done individually in this class. **Store it in**
  `~/private`. 
Secure Shell Connection

- Windows users: Download and install PuTTY
- Connect to Andrew Linux machines using andrew ID and password on port 22.
- Addresses of servers: unix.andrew.cmu.edu
  linux.andrew.cmu.edu
  ghcNN.ghc.andrew.cmu.edu
- *nix and Mac OS users can use the terminal:
  > ssh andrewID@unix.andrew.cmu.edu
Copying files over the network

- If you like, you can also work on your computer and copy files to andrew machines.
- *nix and Mac OS: use `scp`:
  
  ```bash
  > scp local_path
  user@example.address:path_on_remote_host
  > scp user@example.address:path_on_remote_host
  local_path
  ```

- On Windows, PuTTY provides `pscp` which can be used from the command prompt and works the same way.

- Mind your back-slashes and forward-slashes.
Learn ye some emacs and c0

- But first, we need to configure emacs:
  > emacs ~/.emacs

- Append the following lines:
  (setq c0-root "/afs/andrew/course/15/122/")
  (load (concat c0-root "c0-mode/c0.el"))

And let the fun begin!
  > emacs fact.c0
Ye Olde Factorial Functionne: Recursive Definition

1 // What is missing?
2 int fact1(int x)
3 {
4     if (x == 0) {
5         return 1;
6     } else {
7         return x * fact1(x - 1);
8     }
9 }
Factorial not defined for negative numbers!

```c0
1 int fact1(int x)
2    // @requires x >= 0;
3 {
4      if (x == 0) {
5          return 1;
6      } else {
7          return x * fact1(x - 1);
8      }
9  }

> rlwrap coin fact.c0 -d
```
Factorial: An Equivalent Specification

```c
int fact2(int x)
{
    return x == 0 ? 1 : x * fact2(x - 1);
}
```

This uses the **ternary operator**. Compact and useful when the branches of the if-else statement evaluate a single expression each.
What if loops are faster than recursive functions?

This is not necessarily true, but let's implement factorial with loops for the sake of the argument:

```c
int fact3(int x)
{
    int r = 1;
    while (x > 0)
    {
        // @loop_invariant ....;
        r = r * x;
        x--; /* shorthand for x = x - 1 */
    }
    return r;
}
```

Exercise: what is the loop invariant expressions?
Another way of writing it?

```c
int fact4(int x)
{//@requires x >= 0;
//@ensures \result == fact1(x);
{
    int r = 1;

    for (int i = x; i > 0; i--)
        {//@loop_invvariant ....
            {r = r * i;
        }
    }//@assert i == 0;

    return r;
}
```

What is wrong?
Induction variable is out of scope!

The assertion can’t inspect \(i\). Let us fix it:

```c
1 int fact4(int x)
2  // @requires x >= 0;
3  // @ensures \ result == fact1(x);
4 {
5   int r = 1;
6   int i; /* induction variable */
7
8   for (i = x; i > 0; i--)
9      // @loop_invariant . . . ;
10   {
11      r = r * i;
12   }
13  // @assert i == 0;
14
15  return r;
16 }
```
Factorial: Summary

- Four types of contracts: requires, ensures, loop_invariant and assert
- Logically: loop_invariant is a pre-condition and post-condition of the entire loop and each iteration of the loop.
- Operationally: loop_invariant gets checked every time the loop header is evaluated, regardless of whether the test succeeds or fails, and at loop exits.
- Caution: loop_invariant will be checked even if the loop is never entered!
- for loops are idiomatic, but beware of scoping.
- i is called the loop induction variable. Some relation to mathematical induction?
Resources:

▶ http://c0.typesafety.net/
This page has links to:
  ▶ The C0 language reference, if you have questions about syntax, the semantics of operators, the type system, etc.
  ▶ The C0 library reference: this documents functions that we provide (such as console IO and file IO).
  ▶ A C0 tutorial written by friends of the course.

▶ man pages, if you are uncertain of the behavior of shell commands.

▶ office hours:
  ▶ General: Monday and Friday, 3:00-4:20PM, GHC5206
  ▶ Anand <asubrama@andrew.cmu.edu>: Monday, 1:30-2:30, GHC 9th floor kitchenette
  ▶ Kristina <ksojakov@cs.cmu.edu>: Tuesday, 4:20-5:20, GHC 6603