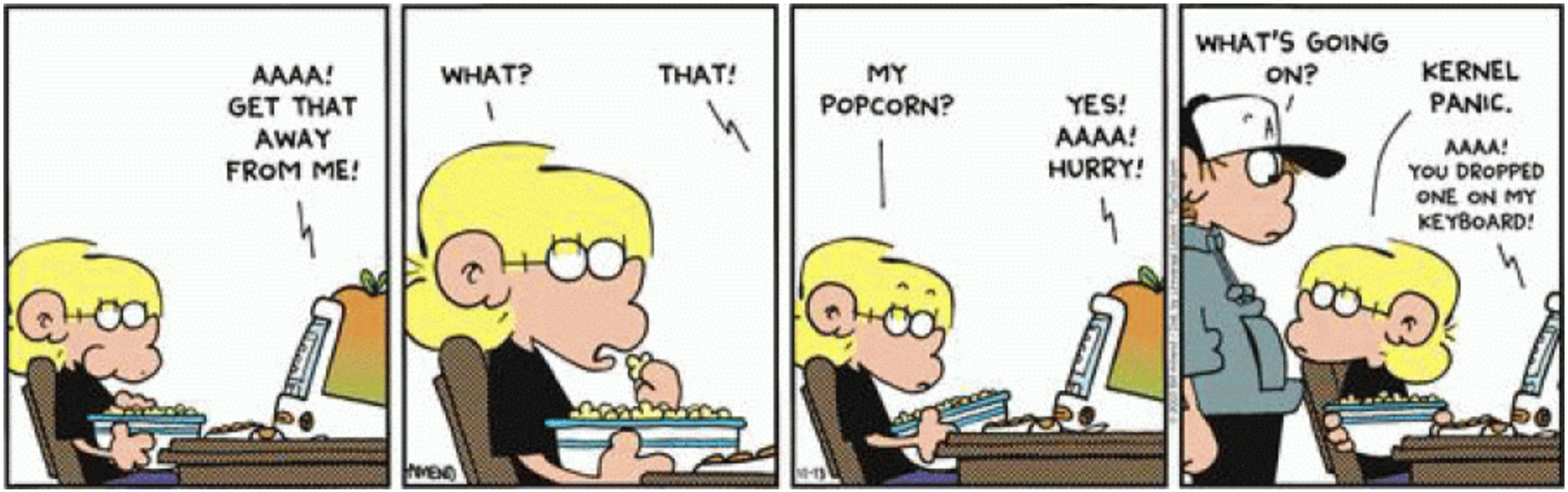
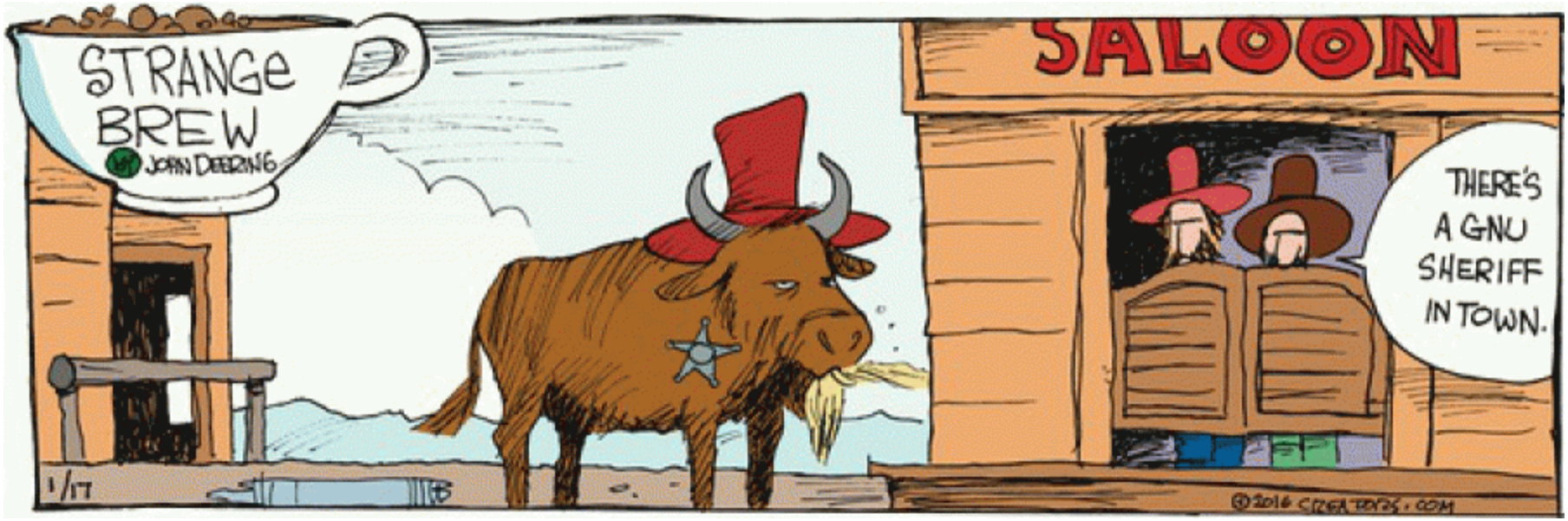
Applied Programming

Basic GNU/Linux Tools


AAAA! GET THAT AWAY FROM ME!
WHAT?
THAT!
MY POPCORN?
YES! AAAA! HURRY!
WHAT'S GOING ON?
KERNEL PANIC.
AAAA! YOU DROPPED ONE ON MY KEYBOARD!


STRANGE BREW
by JOHN DEERING
SALOON
THERE'S A GNU SHERIFF IN TOWN.
1/17
©2016 CREATORS.COM

Basic GNU/Linux Tools

- File names and permissions
  - **`ls`**, **`du`**, **`chmod`**
- Archiving files
  - **`tar`**
- Getting help
  - **`man`**, **`info`**

Unix File Names

- File names can be up to 256 characters long (the following is a 40 character name)

  `xxxxxxxxxx-xxxxxxxxxx_xxxxxxxxxx.xxxxxxxxxxx`

- Valid characters are letters, numbers and 3 special characters: "-", "_", and "."

- Case sensitive

Listing Files and their Attributes

- Use **`ls`** to list files and their attributes
- The most common options are

  `-l` (**`l`**ong)

  `-a` (**`a`**ll, including hiddent "dot files")
- **`ls -l`** displays (in light blue)

Attributes	Links	Owner	Group	Size	Date	Name
drwx------.	1	rhreec	domain users	604	May 21 15:23	**Class**
-rw-r--r--.	1	rhreec	domain users	664	Jun 14 11:46	east.c

- To list filenames and their size:

  `du -sb` (**d**isk **u**sage - **s**ize in **b**ytes)

Attributes Summary

The ***ten characters encoding*** this information are:

type owner group other

d rwx

Think of each as a bit that can be on or off

- **1st character indicates the *type of file***
  - **"-" ordinary file**
  - **"d" directory**
  - **"l" link.**

- **Next three 3 character groups are the permissions**
  - **owner permissions**
  - **group permissions**
  - **other permissions**

Permission Fields

There are ***5 possible*** characters in the ***3 character*** permission fields

```
- = no permission
r = read    - Only the read field.
w = write   - Only the write field.
x = execute - Only in the execute field.
s = setuid  - Only in the execute field.
```

chmod Summary

- chmod [u g o a] [+ -] [r w x] **file**
  - u - user, you
  - g - group, (assigned by the sysadmin)
  - o - other, everyone else in the world
  - a - all - same as ugo, default
  - + - enable
  - - - disable
  - r - read
  - w - write
  - x - execute, e.g allows the file to run as a program
  - file - File or wildcard to apply change to

chmod - Utility to change file permissions

Note: These are the most popular options

Changing Files Attributes

- Make **filename** executable (**a** designates all, e.g., **ugo**)

```
chmod a+x filename
```

- Set permission of **filename** to read and write for all:

```
chmod a+rw filename
chmod  +rw filename        (a is default)
```

- Make all your files private. Deny all permissions for group and other.

```
chmod go-rwx  *
```

The most common use of **chmod** is to "make a file executable"

```
chmod a+x filename
```

Example: Make a bash script executable

Archiving and Listing tarballs

- To archive files we will use **tar** (**ta**pe a**r**chive)

```
tar cvf  myfile.tar *.c        (create)
tar tvf  myfile.tar            (list or test)
tar xvf  myfile.tar            (extract)
```

- To archive and compress (using g**z**ip) files use (not used in our class)

```
tar zcvf  myfile.tgz *.c       (create)
tar ztvf  myfile.tgz           (list)
tar zxvf  myfile.tgz           (extract)
```

- Notes:
  - A tar file is called a "tarball"
  - Omit the verbose option **v** if you don't want the file information echoed (**f** is for file), not recommended.

You will have plenty of opportunities to practice this in the homework

Example 1

- Create files, no details

  tar cf hw1.tar hw1

- Create files, with details (my favorite)

  tar cvf hw1.tar hw1

  hw1/qs
  hw1/myinfo.txt
  hw1/QuadraticSolver.c
  hw1/Makefile
  hw1/screen.jpeg,
  hw1/out.txt
  hw1/analysis.txt
  hw1/.Makefile.swp
  hw1/mem.txt

Example 2

- Extract files, no details

  tar xf hw1.tar

  tar: blocksize = 20

- Extract files, with details (my favorite)

  tar xvf hw1.tar

  tar: blocksize = 20
  x hw1/qs, 11062 bytes, 22 tape blocks
  x hw1/myinfo.txt, 278 bytes, 1 tape block
  x hw1/QuadraticSolver.c, 2924 bytes, 6 tape blocks
  x hw1/Makefile, 1429 bytes, 3 tape blocks
  x hw1/screen.jpeg, 127397 bytes, 249 tape blocks
  x hw1/out.txt, 404 bytes, 1 tape block
  x hw1/analysis.txt, 2820 bytes, 6 tape blocks
  x hw1/.Makefile.swp, 12288 bytes, 24 tape blocks
  x hw1/mem.txt, 640 bytes, 2 tape blocks

Example 3

- List the tar file, no details (another favorite)

```
tar  tf  hw1.tar
hw1/
hw1/qs
hw1/myinfo.txt
hw1/QuadraticSolver.c
hw1/Makefile
hw1/screen.jpeg
hw1/out.txt
hw1/analysis.txt
hw1/.Makefile.swp
hw1/mem.txt
```

- List the tar file, full details

```
tar  tvf  hw1.tar
drwxr-xr-x  1 31393  3128        0 Feb  4 2016 hw1/
-rwxr-xr-x  1 31393  3128    11062 Feb  4 2016 hw1/qs
-rw-r--r--  1 31393  3128      278 Feb  3 2016 hw1/myinfo.txt
-rw-r--r--  1 31393  3128     2924 Feb  3 2016 hw1/QuadraticSolver.c
-rw-r--r--  1 31393  3128     1429 Feb  4 2016 hw1/Makefile
-rw-r--r--  1 31393  3128   127397 Feb  3 2016 hw1/screen.jpeg
-rw-r--r--  1 31393  3128      404 Feb  4 2016 hw1/out.txt
-rw-r--r--  1 31393  3128     2820 Feb  4 2016 hw1/analysis.txt
-rw-r--r--  1 31393  3128    12288 Feb  4 2016 hw1/.Makefile.swp
-rw-r--r--  1 31393  3128      640 Feb  4 2016 hw1/mem.txt
```

I/O Redirection

- Any program taking input from the standard input (*e.g.* keyboard) can be redirected to take input from any input file.

```
myprogram < input_file
```

- Any program writing to standard output (*e.g.*, screen) can be redirected to write to any output file

```
myprogram > output_file
```

We will use this in our programming assignments

I/O Redirection

- We can also combine input and output

```
myprogram < input_file > output_file
```

- Create or append options
  - `out_file > in_file (created)`
  - `out_file >> in_file (appended)`

  - `in_file (created) < out_file`
  - `in_file (appended) << out_file`

I/O Redirection

- Redirect **`stderr`** to a file
  - keep an error log

```
MyProgram  2>error.log
```

- Redirect both **`stderr`** and **`stdout`** to a file (no screen output)
  - Note the absence of file descriptors.

```
MyProgram  &>output.log
```

Pipes

- Standard output from one program can be "piped" (symbol "|") to the standard input of another program.
  - Linux programmers use this feature to build modular programs that can "chain"
  - We will use this in HW 8.

- E.g.:
  - ls –la - full list of files
  - ls –la | sort - full list of files sorted alphabetically

http://2.wlimg.com/product_images/bc-full/dir_109/3261561/watermark/oil-gas-line-pipes-3pe-3pp-1481114.jpg

display

- The Linux command to display images and graphs
  - E.g.: display image.gif
- Requires Xwindows

cat

- The Linux command to print the contents of text files to the console
  - E.g.: cat analysis.txt
    - Prints analysis.txt to the screen

Other commands

- rm [-r]- remove files or directories
  - -r - optional recursion flag
  - rm *.txt - remove all text files
  - rm **–r** hw1 - remove hw1 and all the files in it

- cd – change directory
  - cd hw1 - change directory down to “hw1”
  - cd .. - go back up one directory

Getting Help in GNU/Linux

- The classical UNIX way (manual pages)
  - **`man subject`**
  - **`man -S section subject`**
    - **`man -S 3 float`** (section **`3`** is for C)
  - **`man -k keyword`**
- The GNU/Linux way (info reader)
  - **`info subject`**
  - **`info -k keyword`**

Q: find out how to use the C function **`printf`**

Editors

- **vi** (and **vim**)**
  - Very powerful, steep learning curve.
  - Many tutorials available online, e.g., see "Vi Lovers Home Page" for links

    http://thomer.com/vi/vi.html
  - If you have never used it try this interactive tutorial (have a taste of it):

    http://www.openvim.com/tutorial.html

Enable spell checking "setlocal spell spelllang=en_us"

**I use this one

Editors

- **nano**
  - Very easy to use and learn.
  - Sample**.nanorc** with useful settings for programming:

```
include "/usr/share/nano/c.nanorc"
include "/usr/share/nano/python.nanorc"
include "/usr/share/nano/sh.nanorc"
set autoindent
set multibuffer
```

  - Call it with **–c** to show ***line/col of cursor***

    (hint define alias: alias nano = 'nano –c')

C Complier

- **gcc (GNU C Compiler)**
  - The "common denominator of all C compilers"
  - *faithfully implements* the **C** language
    - Use **–std=c89** (**c99** or **c11**) to enforce ANSI C89, C99 or C11, respectively
    - Use –ansi to force ANSI C89

- General invocation
  - **gcc <option flags> <file list>**

- More details later
  - FYI: **C++ is NOT a C compiler!**

Exercise 1

- What TAR command do you use to verify the contents of your TAR file before you submit it?

- **`tar tvf  myfile.tar`** or
- **`tar tf  myfile.tar`**

Exercise 2

- Your professor provided a “binary” in the homework tar file but it won’t execute, what do you do?

- **`Use the chmod command to set the binary to executable`**

```
chmod +x binary
```

Exercise 3

- Your professor wants the output from:
  - 1) STDIO written to a text file
  - 2) STDERR written to a text file
  - 3) Both written to the same file at the same time

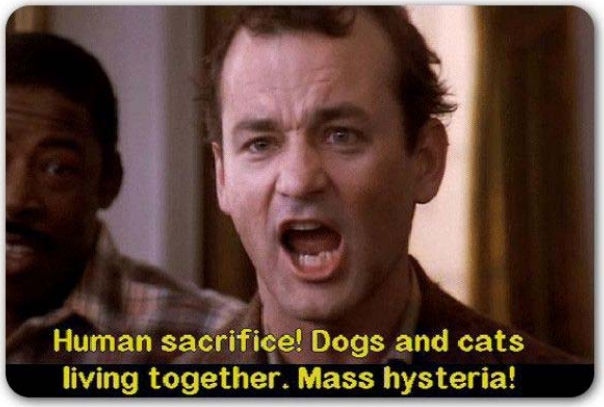
- Use redirection
  - *1) MyProgram* > msg.log
  - 2) *MyProgram* 2> error.log
  - 3) *MyProgram* &> both.log OR
    MyProgram > msg.log 2>&1

- *Note: MyProgram* 2>&1 > msg.log won't work!

Exercise 4

- What happens if you execute the command:

  tar cf *



- Bad things! "*" returns the list of files

  e.g."Darray.c other.c test.c "

- So tar really sees:

  tar cf Darray.c other.c test.c

  – **Destroys Darray.c** and then puts the rest of the files into a new tar file called: Darray.c

    - **Tar files can have ANY name**

- Always **verify** your tar file has a type of **".tar"**

https://s-media-cache-ak0.pinimg.com/736x/d0/f3/24/d0f324c6e5dfe67f93a2d2c0f147f27c.jpg

Example Standard Output

```
/* Simple stdout and stderr code */

#include <stdio.h>
#include <stdlib.h>

int main(int argv, char *argc[]) {
  fprintf(stderr, "Std err\n");
  fprintf(stdout, "Std out\n");
  return 0;
}
```

Required Reading – next class

- Download from MyCourses

  C for Java Programmers – Maassen.pdf