
CSE 390a

Lecture 1

introduction to Linux/Unix environment

slides created by Marty Stepp, modified by Jessica Miller & Ruth Anderson

<http://www.cs.washington.edu/390a/>

Lecture summary

- Course introduction and syllabus
- Unix and Linux operating system
- Introduction to Bash shell

Course Staff

- Me:
 - Ruth Anderson, rea@cs
 - Office hours in CSE 460:
 - Mon 2:30-3:30pm,
 - Tues 10:30-11:30,
 - and by appointment

Course Introduction

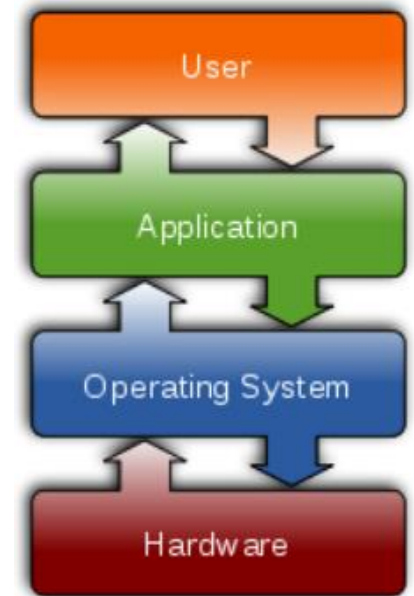
- CSE390a
 - Collection of tools and topics not specifically addressed in other courses that CSE majors should know
 - CSE 351 may be the first course you take that uses Linux
 - Course Topics: Linux command line interface (CLI), Shell scripting, compilation tools (makefiles), version control...
 - Credit / No Credit course, determined by short weekly assignments and a “final” assignment

Operating systems

- What is an OS? Why have one?
- What is a Kernel?

Operating systems

- **operating system:** Manages activities and resources of a computer.
 - software that acts as an interface between hardware and user
 - provides a layer of abstraction for application developers
- features provided by an operating system:
 - ability to execute programs (and multi-tasking)
 - memory management (and virtual memory)
 - file systems, disk and network access
 - an interface to communicate with hardware
 - a user interface (often graphical)
- **kernel:** The lowest-level core of an operating system.



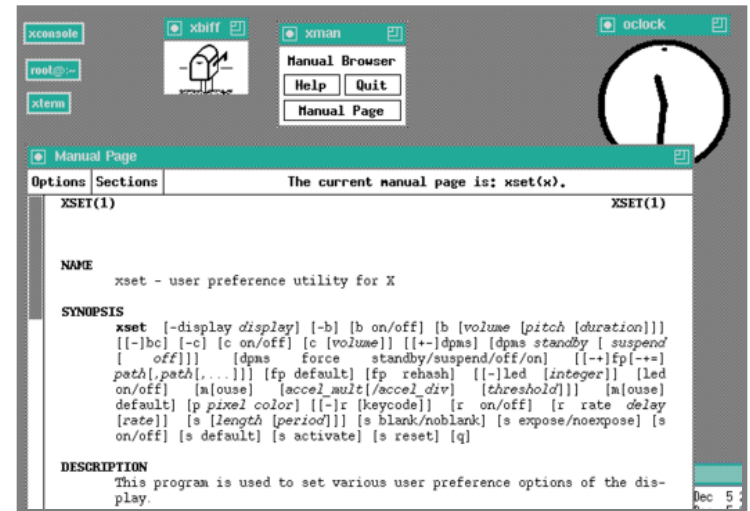
Unix

- brief history:

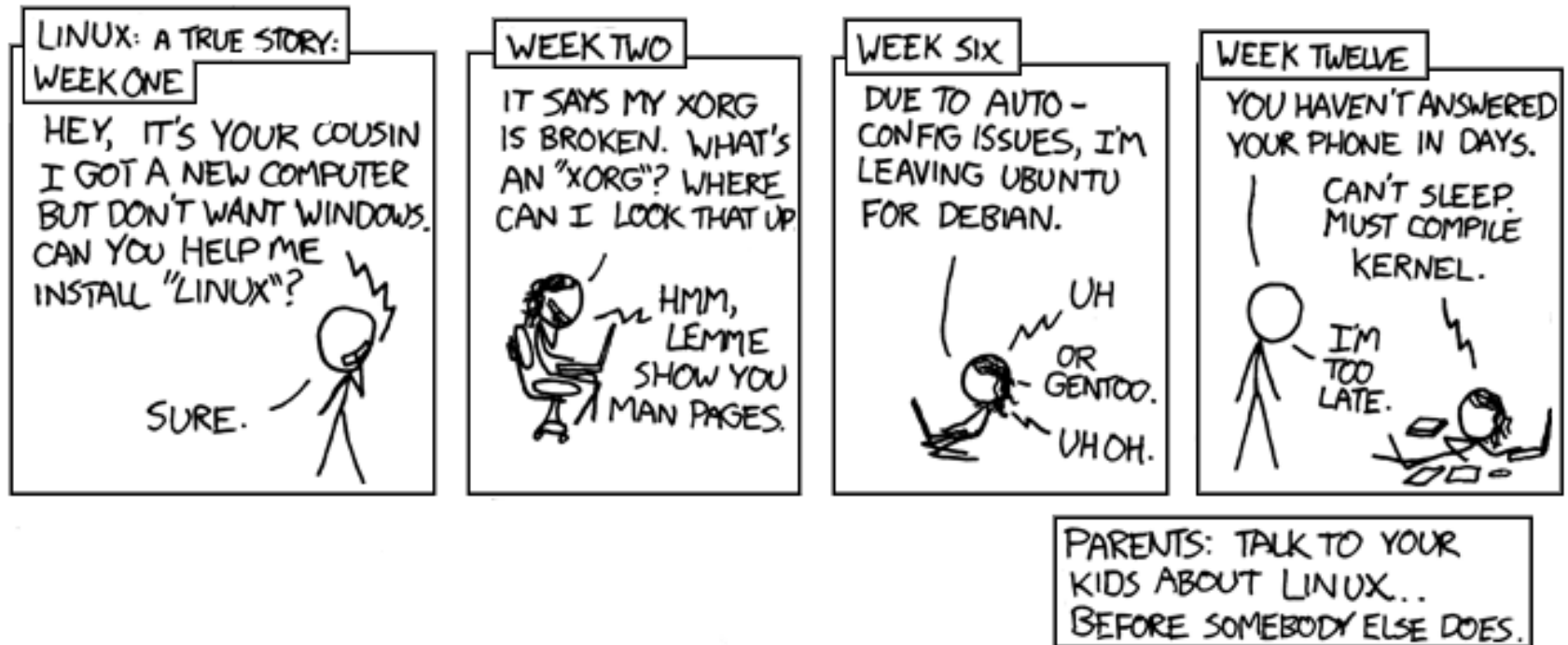
- Multics (1964) for mainframes
- Unix (1969)
- K&R
- Linus Torvalds and Linux (1992)

- key Unix ideas:

- written in a high-level language (C)
- virtual memory
- hierarchical file system; "everything" is a file
- lots of small programs that work together to solve larger problems
- security, users, access, and groups
- human-readable documentation included



On to Linux



Courtesy XKCD.com

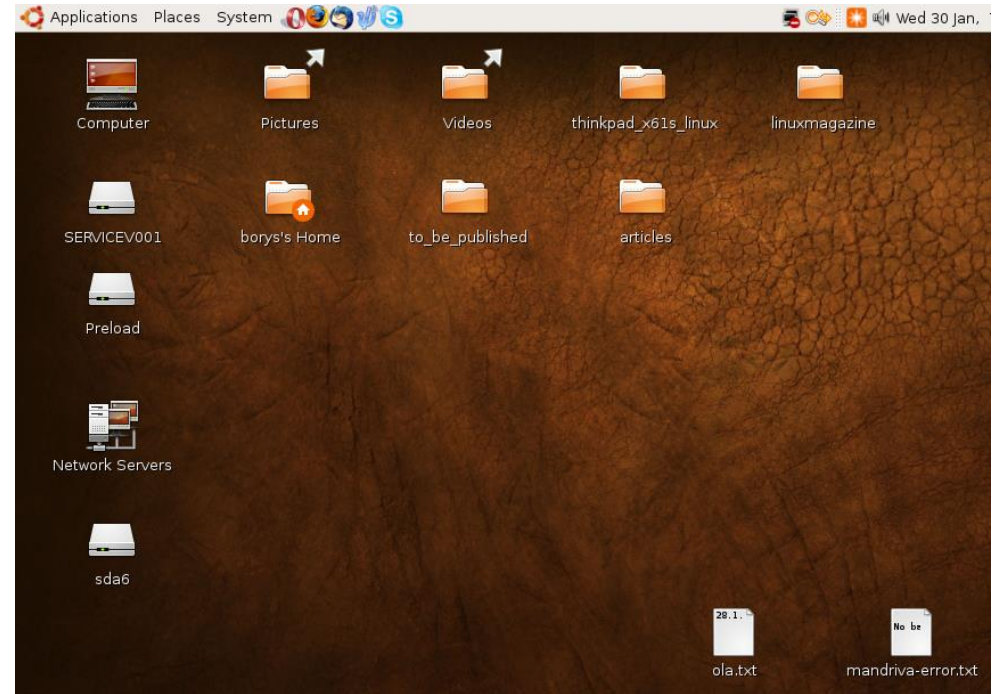
Linux

- **Linux:** A kernel for a Unix-like operating system.
 - commonly seen/used today in servers, mobile/embedded devices, ...
- **GNU:** A "free software" implementation of many Unix-like tools
 - many GNU tools are distributed with the Linux kernel
- **distribution:** A pre-packaged set of Linux software.
 - examples: Ubuntu, Fedora
- key features of Linux:
 - **open source software:** source can be downloaded
 - free to use
 - constantly being improved/updated by the community



Linux Desktop

- X-windows
- window managers
- desktop environments
 - Gnome
 - KDE
- How can I try out Linux?
 - CSE Virtual machine
 - CSE basement labs
 - attu shared server



Things you can do in Linux

- Load the course web site in a browser
- Install and play games
- Play MP3s
- Edit photos
- IM, Skype

Shell

- **shell**: An interactive program that uses user input to manage the execution of other programs.
 - A command processor, typically runs in a text window.
 - User types commands, the shell runs the commands
 - Several different shell programs exist:
 - bash : the default shell program on most Linux/Unix systems
 - We will use bash
 - Other shells: Bourne, csh, tsch
- Why should I learn to use a shell when GUIs exist?

Why use a shell?

- Why should I learn to use a shell when GUIs exist?
 - faster
 - work remotely
 - programmable
 - customizable
 - repeatable

Shell commands

command	description
exit	logs out of the shell
ls	lists files in a directory
pwd	outputs the current working directory
cd	changes the working directory
man	brings up the manual for a command

```
$ pwd
/homes/iws/rea
$ cd CSE390
$ ls
file1.txt file2.txt
$ ls -l
-rw-r--r-- 1 rea    fac_cs 0 2012-03-29 17:45 file1.txt
-rw-r--r-- 1 rea    fac_cs 0 2012-03-29 17:45 file2.txt
$ cd ..
$ man ls
$ exit
```

Relative directories

directory	description
.	the directory you are in ("working directory")
..	the parent of the working directory (../.. is grandparent, etc.)
~	your home directory (on many systems, this is /home/ <i>username</i>)
~ <i>username</i>	<i>username</i> 's home directory
~/Desktop	your desktop

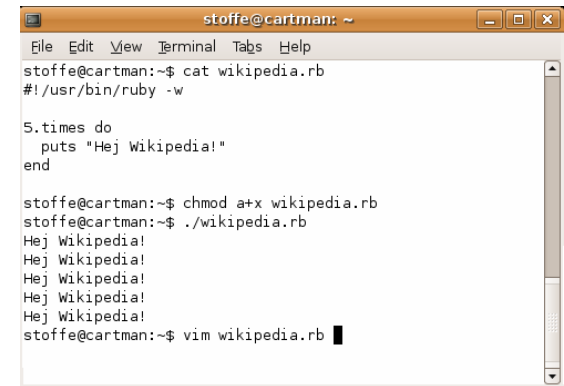
Directory commands

command	description
<code>ls</code>	list files in a directory
<code>pwd</code>	output the current working directory
<code>cd</code>	change the working directory
<code>mkdir</code>	create a new directory
<code>rmdir</code>	delete a directory (must be empty)

- some commands (`cd`, `exit`) are part of the shell ("builtins")
- others (`ls`, `mkdir`) are separate programs the shell runs

Shell commands

- many accept **arguments** or **parameters**
 - example: cp (copy) accepts a source and destination file path
- a program uses 3 streams of information:
 - stdin, stdout, stderr (standard in, out, error)
- **input**: comes from user's keyboard
- **output**: goes to console
- **errors** can also be printed (by default, sent to console like output)
- parameters vs. input
 - *parameters*: before Enter is pressed; sent in by shell
 - *input*: after Enter is pressed; sent in by user



```
stoffe@cartman: ~  
File Edit View Terminal Tabs Help  
stoffe@cartman:~$ cat wikipedia.rb  
#!/usr/bin/ruby -w  
  
5.times do  
  puts "Hej Wikipedia!"  
end  
  
stoffe@cartman:~$ chmod a+x wikipedia.rb  
stoffe@cartman:~$ ./wikipedia.rb  
Hej Wikipedia!  
Hej Wikipedia!  
Hej Wikipedia!  
Hej Wikipedia!  
Hej Wikipedia!  
Hej Wikipedia!  
stoffe@cartman:~$ vim wikipedia.rb
```

Command-line arguments

- most options are a - followed by a letter such as -c
 - some are longer words preceded by two - signs, such as --count
- options can be combined: `ls -l -a -r` can be `ls -lar`
- many programs accept a --help or -help option to give more information about that command (in addition to man pages)
 - or if you run the program with no arguments, it may print help info
- for many commands that accept a file name argument, if you omit the parameter, it will read from standard input (your keyboard)

Shell/system commands

command	description
man or info	get help on a command
clear	clears out the output from the console
exit	exits and logs out of the shell

command	description
date	output the system date
cal	output a text calendar
uname	print information about the current system

- "man pages" are a very important way to learn new commands
man ls
man man

File commands

command	description
cp	copy a file
mv	move or rename a file
rm	delete a file
touch	create a new empty file, or update its last-modified time stamp

- caution: the above commands do not prompt for confirmation
 - easy to overwrite/delete a file; this setting can be overridden (how?)
- *Exercise* : Given several albums of .mp3 files all in one folder, move them into separate folders by artist.
- *Exercise* : Modify a .java file to make it seem as though you finished writing it on Dec 28 at 4:56am.

Mounting cse homedir on VM

<https://www.cs.washington.edu/lab/software/homeVMs/linuxVM#install>

- Create a directory in your home directory, called csehomedir:
 - `cd`
 - `mkdir csehomedir`
- Now to use that directory as a “link” to your CSE files on your VM:
 - `sshfs username@attu: ~/csehomedir` **OR**
 - `sshfs username@attu.cs.washington.edu:/homes/iws/username ~/csehomedir/`
- It is a good idea to back up your files from your VM regularly.
 - Actually keep your files on your CSE home directory
 - Regularly move files from your VM to another location
 - If you need to get a fresh VM image, you can save the files from your old VM using this procedure: **"My VM Seems Broken. How Do I Recover?"**
- <https://www.cs.washington.edu/lab/software/homeVMs/linuxVM#faq>

My VM is Broken!

<https://www.cs.washington.edu/lab/software/homeVMs/linuxVM#install>

- If your VM is misbehaving, first try a reboot of the VM and also of your machine. If that doesn't work, often it is easiest just to get a fresh VM image and start over (maybe you saved the .zip file you downloaded previously?)
- BEFORE you delete your current copy of the VM, you can save the files from your current copy of the VM using this procedure:
 - See "My VM Seems Broken. How Do I Recover?" here:
<https://www.cs.washington.edu/lab/software/homeVMs/linuxVM#faq>

Exercise Solutions

- caution: the `cp`, `rm`, `mv` commands do not prompt for confirmation
 - easy to overwrite/delete a file; this setting can be overridden (how?)
 - Use “-i” with the command, “interactive” to prompt before overwrite
- *Exercise* : Given several albums of `.mp3` files all in one folder, move them into separate folders by artist.
 - `mkdir U2`
 - `mkdir PSY`
 - `mkdir JustinBieber`
 - `mv GangnamStyle.mp3 PSY/`
 - `mv Pride.mp3 U2/`
- *Exercise* : Modify a `.java` file to make it seem as though you finished writing it on Dec 28 at 4:56am.
 - `touch -t 201412280456 Hello.java`

Basic Emacs Commands

- C- = control key M- = meta/alt key
- read a file into Emacs: C-x C-f
- save a file back to disk: C-x C-s
- exit Emacs permanently: C-x C-c
- search forward: C-s search backward: C-r
- scroll to next screen: C-v scroll to previous screen: M-v
- Undo: C-x u

entity to move over	backward	forward
character	C-b	C-f
word	M-b	M-f
line	C-p	C-n
go to line beginning/end	C-a	C-e
go to buffer beginning/end	M-<	M->

Basic Vim Commands

- `:w` Write the current file
- `:wq` Write the current file and exit.
- `:q!` Quit without writing
- To change into insert mode: `i` or `a`
 - Use escape to exit
- search forward `/`, repeat the search backwards: `N`
- Basic movement:
 - `h l k j` character left, right; line up, down (also arrow keys)
 - `b w` word/token left, right
 - `ge e` end of word/token left, right
 - `0 $` jump to first/last character on the line
- `x` delete
- `u` undo