CS35L Lab4

Spring 2017

Introduction about myself

Sophia Yan

- Master student in CS @ UCLA
- B.S. in Math & CS @ UIUC

-TA for CS35L Lab4

-sophiaxueting@gmail.com

-Office hour: TBA

Introduction to the course

- 2 unit course, lab-oriented
- Instructor Prof. Paul Eggert
- Prerequisite CS 31
- Course website
  http://web.cs.ucla.edu/classes/spring17/cs35L/
- Piazza (https://piazza.com)
- Use SEASnet account to log in server
- No attendance taken
- Discussion is encouraged, but final work should be done individually

- **Grading**
  - **Assignments - 50%** (equally weighted)
  - **Final exam - 50%**
- 10 assignments
  - 9 regular assignments + 1 presentation
- Regular assignments
  - Lab exercises (expected to be done in the lab)
  - Homework
  - Due every week Friday 11:55pm
- Presentation (Assignment 10)
  - Will distribute sign up sheet
  - 10 mins presentation + research report
  - Due the end of week 10

Week 1	Introduction to linux
Week 2	Shell scripting
Week 3	Modifying and rewriting softwares (Python)
Week 4	C programming and debugging
Week 5	System call programming and debugging (with C)
Week 6	Multithreaded performance (with C)
Week 7	SSH
Week 8	Dynamic linking
Week 9	Change management (git)
Week 10	Research presentation

- All assignments to be done individually
- Submitted on CCLE
- Lateness penalty (with some exceptions)
  - 2^N % deduction for being up to N days late
  - Exception 1: Last assignment must be submitted on time
  - Exception 2: Not accepting submissions after last day of instruction
  - Other exceptions need approval from Prof. Eggert

- Final
  - 11:30 AM - 2:30 PM, Thursday, June 15, 2017
  - Different cross sections
  - Open-book, open-notes, no electronic device
  - All materials will be covered, including concept, programming and presentations

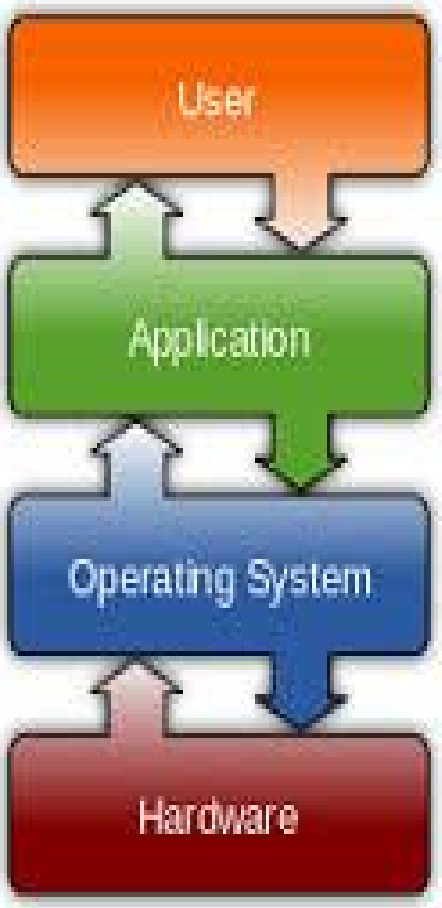
Introduction to Linux

Week 1

What is Linux?

- Operating system !
- Created by Linus and a group of people (online)
- Unix-like open source software
- Free to contribute, free to use

- Four Components (linux distribution)
  - Linux kernel
  - GNU utilities
  - Graphical desktop environment
  - Application software



Linux Kernel

- Four main functionalities
  - System memory management
  - Software program management
  - Hardware management
  - Filesystem management

GNU utilities

- System utilities to run on linux kernel
- Contains coreutils package
  - Handling files
  - Manipulating text
  - Managing text
- Shell is a special interactive utility (CLI)
  - Bash is the default shell in Linux

Graphical desktop environment

- Two common graphical environment
  - KDE
  - GNOME desktop

A Brief History of Operating Systems

- The Dark Ages
  - No OS until 1960s
  - Manually loaded programs
  - Reboot after each program
- Batch OS
  - Unified application development across systems
  - Output via printer, later via monitor
  - I/O via magnetic tape or disk
  - Written in assembler (e.g., OS/360)
  - Multiprocess

A Brief History of Operating Systems

- Timesharing OS
  - Multiuser
  - Multics (1964)
    - Segmented memory
    - Paged virtual memory
    - Applications written in many languages
    - Shared multiprocess memory
- Personal Computer
  - Single machine for single user
  - OS must manage screen and input devices
  - Window, Icon, Menu, Pointing Device (WIMP, e.g., MacOS, 1984)
- Cutting-Edge OS
  - High performance computer (HPC) clusters (e.g., BlueGene/L at LLNL rated at 280.6 teraFLOPS)
  - Cell phones, video
  - Video games
  - Browsers

GUI – Graphic User Interface

- Human-computer interface using graphic icons and visual indicators
- Intuitive
- Limited Control
- Easy multitasking
- Limited by pointing
- Bulky remote access
- Example GUI in Linux : X (Windows), Gnome, KDE (Linux)

CLI – Command Line Interface

- Human-computer interface using solely text input and output
- Pure control (e.g., scripting)
- Cumbersome multitasking
- Speed: Hack away at keys
- Convenient remote access (e.g. ssh)
- Example CLI: bash (linux), xterm (Windows)

Unix File System Layout

- A file system used by many Unix and Unix-like operating systems, including Linux
- Everything is a file (including devices)
- Tree structured hierarchy (with some exceptions)

Demo

- Log in with your Seasnet account
- Use Putty or ugrad or lnxsrv server
- Lost?
  - man <command>
  - Look up command usage in manual pages!
  - https://www.tutorialspoint.com/unix/unix-file-system.htm

The Basics: Moving Around

- pwd: print working directory
- cd: change working directory
- ~: home directory
- .: current directory
- /: root directory, or directory separator
- ..: parent directory

The Basics: Dealing with Files

- The basics continued…
  - mv: move a file (no undos!)
  - cp: copy a file
  - rm: remove a file
  - mkdir: make a directory
  - rmdir: remove a directory
  - ls: list contents of a directory
    - -a: list all files including hidden ones
    - -l: show long listing including permission info
    - -s: show size of each file, in blocks

The Basics: Look These Up

- cat
- head
- tail
- du
- ps
- kill
- diff
- cmp
- wc
- sort

The Basics: Redirection

- > *file*: write stdout to a file
- >> *file*: append stdout to a file
- < *file*: use contents of a file as stdin

The Basics: wh.. command

- whatis <command>
  - return name section of man page
- whereis <command>
  - locates the binary, source and man page files for a command
- which <command>
  - locate a program file in the user's path

Q: difference between whereis and which?

Format of submission

- For lab questions(ans1.txt)
  - Answer 15 questions using natural language
  - List all the commands used to solve the problem
  - Give some explanations about your choice of commands
  - No need for keystrokes in this file
  - Will be graded manually

CS35L Lab4

Spring 2017

Administration

- Instructor's office hour:

  Mondays 14:00–15:00 and Thursdays 09:30–10:30

  @BH 4532J

- Assignment 1 is due Saturday 4/8 11:55pm on CCLE
- http://web.cs.ucla.edu/classes/spring17/cs35L/assign.html
- Two files: ans1.txt, key1.txt -- manually graded

Work using your own computer

- Mac user

- Download ssh client

- ssh accountname@ugrad.seas.ucla.edu

- Windows user

- Download PUTTY

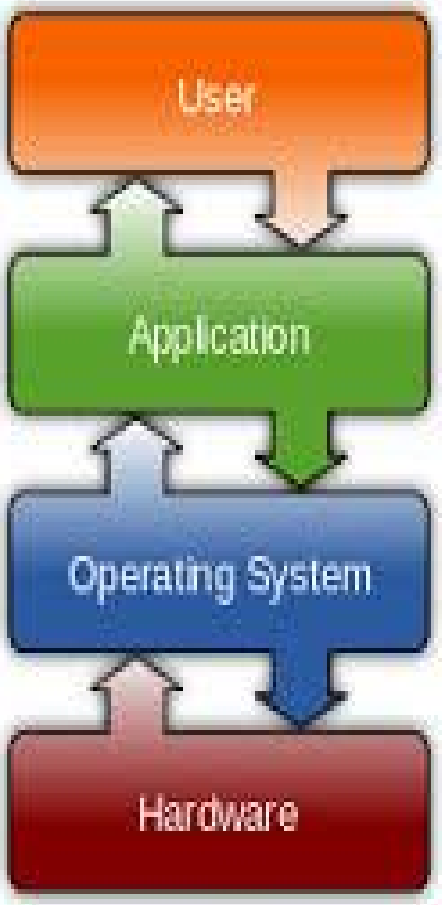
- Do similar things as above

- Make sure your code works on the server before submitting!

What is Linux?

- Operating system !
- Created by Linus and a group of people (online)
- Unix-like open source software
- Free to contribute, free to use

- Four Components (linux distribution)
  - Linux kernel
  - GNU utilities
  - Graphical desktop environment
  - Application software



Linux Kernel

- Four main functionalities
  - System memory management
  - Software program management
  - Hardware management
  - Filesystem management

GNU utilities

- System utilities to run on linux kernel
- Contains coreutils package
  - Handling files
  - Manipulating text
  - Managing text
- Shell is a special interactive utility (CLI)
  - Bash is the default shell in Linux

Graphical desktop environment

- Two common graphical environment
  - KDE
  - GNOME desktop

Linux File System Layout

- Stores files in a single virtual directory
  - Does not use drive letters in pathnames
- Tree structured hierarchy
- Everything is a file (including devices)
- Each user has a separate home directory
  - A new shell starts from home directory

Common Linux Directory Names

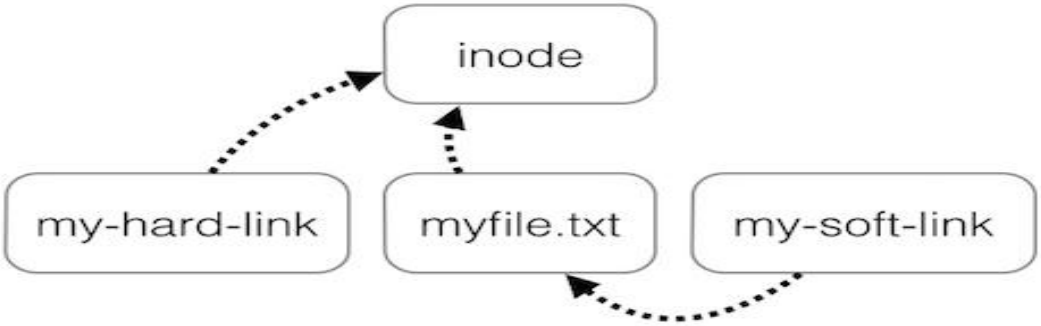
Directory	Usage
/	The root of the virtual directory. Normally, no files are placed here.
/bin	The binary directory, where many GNU user-level utilities are stored.
/boot	The boot directory, where boot files are stored.
/dev	The device directory, where Linux creates device nodes.
/etc	The system configuration files directory.
/home	The home directory, where Linux creates user directories.
/lib	The library directory, where system and application library files are stored.
/media	The media directory, a common place for mount points used for removable media.
/mnt	The mount directory, another common place for mount points used for removable media.
/opt	The optional directory, often used to store optional software packages.
/root	The root home directory.
/sbin	The system binary directory, where many GNU admin-level utilities are stored.
/tmp	The temporary directory, where temporary work files can be created and destroyed.
/usr	The user-installed software directory.
/var	The variable directory, for files that change frequently, such as log files.

Path in linux file system

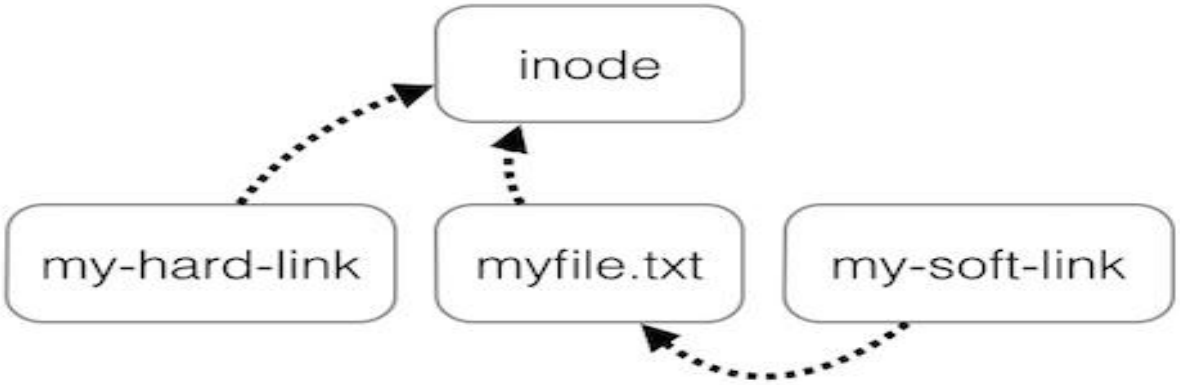
- Absolute path
  - Path from root directory
- Relative path
  - Path relative to current directory

File

- touch filename
  - Create an empty file
  - Update the modification and access time of file
- File is represented as inode (index node)
  - Each inode has an inode number
  - ls -i
- File != File name
  - Many file names can refer to the same copy of data

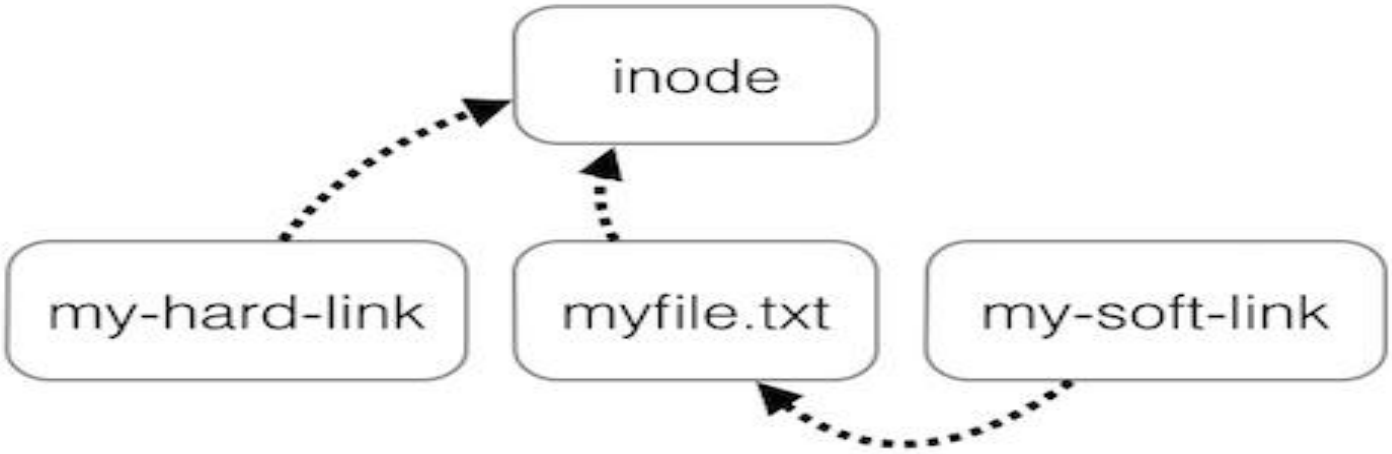


- Two types of links
  - Hard links: points to physical data (inode)
  - Soft/symbolic link (-s): points to a file
- ln *file filename*
  - create a hard link by default
  - option -s creates soft link
- readlink *filename*
  - Resolve symbolic link



Question:

- Does cp command create links or copy data?
- What is the potential problem of running rm command in the case?



File Attributes

- File attribute: many (list some?)
- stat: display file status

```
  File: 'test.txt'
  Size: 22              Blocks: 0          IO Block: 65536  regular file
Device: 1fh/31d Inode: 4910057     Links: 1
Access: (0644/-rw-r--r--)  Uid: (10422/ cs35lt4)   Gid: (  300/ taaccts)
Access: 2017-04-05 23:00:11.047000000 -0700
Modify: 2017-04-05 23:00:11.047996000 -0700
Change: 2017-04-05 23:00:11.048995000 -0700
 Birth: -
```

- file: check file type
- Text files
- Executable files
- Data files

File Attributes

Permission: limit of access for different users

- 3 users : user (u), group (g), others (o)
- 3 permission: read (r, 4), write (w, 2),executable (x, 1)

- chmod
  - change file modes
  - symbolic mode: chmod u+x *filename*
  - numerical mode: chmod 751 *filename*

Search files in a directory

- find
- Walk down a file hierarchy
- Options
  - type: type of a file (directory, symbolic link)
  - perm: permission of a file
  - name: name of a file
  - prune: don’t descend into a directory
  - ls: list current file

File Name Matching in option name

- ?: matches any single character in a filename
- *: matches zero or more characters in a filename
- []*:* matches any one of the characters between the brackets. Use '-' to separate a range of consecutive characters.

The Basics: History

- <up arrow>: previous command
- <tab>: auto-complete
- !!: replace with previous command
- ![*str*]: refer to previous command with str
- ^[*str*]: replace with command referred to as str

vi

- Modes:
  - Normal: Enter commands
  - Insert: Insert text
  - Visual: Like normal, but you can highlight
  - Replace: Like insert, but you replace characters as you type
  - Recording: Record a sequence of key sequences

VI "Cheat" Sheet
ACNS Bulletin ED–03
February 1995

vi Editor "Cheat Sheet"

Invoking vi: vi *filename*

Format of vi commands: *[count][command]* (count repeats the effect of the command)

Command mode versus input mode

Vi starts in command mode. The positioning commands operate only while vi is in command mode. You switch vi to input mode by entering any one of several vi input commands. (See next section.) Once in input mode, any character you type is taken to be text and is added to the file. You cannot execute any commands until you exit input mode. To exit input mode, press the escape (**Esc**) key.

Input commands (end with Esc)

a	Append after cursor
i	Insert before cursor
o	Open line below
O	Open line above
:r *file*	Insert *file* after current line

Any of these commands leaves vi in input mode until you press **Esc**. Pressing the **RETURN** key will not take you out of input mode.

Change commands (Input mode)

cw	Change word (Esc)
cc	Change line (Esc) - blanks line
c$	Change to end of line
r*c*	Replace character with *c*
R	Replace (Esc) - typeover
s	Substitute (Esc) - 1 char with string
S	Substitute (Esc) - Rest of line with text
.	Repeat last change

Changes during insert mode

<ctrl>h	Back one character
<ctrl>w	Back one word
<ctrl>u	Back to beginning of insert

File management commands

:w *name*	Write edit buffer to file *name*
:wq	Write to file and quit
:q!	Quit without saving changes
ZZ	Same as :wq
:sh	Execute shell commands (<ctrl>d)

Window motions

<ctrl>d	Scroll down (half a screen)
<ctrl>u	Scroll up (half a screen)
<ctrl>f	Page forward
<ctrl>b	Page backward
/string	Search forward
?string	Search backward
<ctrl>l	Redraw screen
<ctrl>g	Display current line number and file information
n	Repeat search
N	Repeat search reverse
G	Go to last line
*n*G	Go to line *n*
:*n*	Go to line *n*
z<CR>	Reposition window: cursor at top
z.	Reposition window: cursor in middle
z-	Reposition window: cursor at bottom

Cursor motions

H	Upper left corner (home)
M	Middle line
L	Lower left corner
h	Back a character
j	Down a line
k	Up a line
^	Beginning of line
$	End of line
l	Forward a character
w	One word forward
b	Back one word
f*c*	Find *c*
;	Repeat find (find next *c*)

Deletion commands

dd or *n*dd	Delete *n* lines to general buffer
dw	Delete word to general buffer
d*n*w	Delete *n* words
d)	Delete to end of sentence
db	Delete previous word
D	Delete to end of line
x	Delete character

Recovering deletions

p	Put general buffer after cursor
P	Put general buffer before cursor

Undo commands

u	Undo last change
U	Undo all changes on line

Rearrangement commands

yy or Y	Yank (copy) line to general buffer
"*a*6yy	Yank 6 lines to buffer *a*
yw	Yank word to general buffer
"*a*9dd	Delete 9 lines to buffer *a*
"A9dd	Delete 9 lines; Append to buffer *a*
"*a*p	Put text from buffer *a* after cursor
p	Put general buffer after cursor
P	Put general buffer before cursor
J	Join lines

Parameters

:set list	Show invisible characters
:set nolist	Don't show invisible characters
:set number	Show line numbers
:set nonumber	Don't show line numbers
:set autoindent	Indent after carriage return
:set noautoindent	Turn off autoindent
:set showmatch	Show matching sets of parentheses as they are typed
:set noshowmatch	Turn off showmatch
:set showmode	Display mode on last line of screen
:set noshowmode	Turn off showmode
:set all	Show values of all possible parameters

Move text from file *old* to file *new*

vi *old*	
"*a*10yy	yank 10 lines to buffer *a*
:w	write work buffer
:e *new*	edit new file
"*a*p	put text from *a* after cursor
:30,60w *new*	Write lines 30 to 60 in file *new*

Regular expressions (search strings)

^	Matches beginning of line
$	Matches end of line
.	Matches any single character
*	Matches any previous character
.*	Matches any character

Search and replace commands

Syntax:

```
:[address]s/old_text/new_text/
```

Address components:

.	Current line
n	Line number n
.+m	Current line plus m lines
$	Last line
/string/	A line that contains "string"
%	Entire file
[addr1],[addr2]	Specifies a range

Examples:

The following example replaces only the first occurrence of Banana with Kumquat in each of 11 lines starting with the current line (.) and continuing for the 10 that follow (.+10).

```
:.,.+10s/Banana/Kumquat
```

The following example replaces every occurrence (caused by the `g` at the end of the command) of `apple` with `pear`.

```
:%s/apple/pear/g
```

The following example removes the last character from every line in the file. Use it if every line in the file ends with `^M` as the result of a file transfer. Execute it when the cursor is on the first line of the file.

```
:%s/.$//
```

GNU Emacs

- A free, portable, extensible text editor
- More powerful than a text editor
  - A programming environment
  - Can compile, evaluate, debug etc.
- Only one mode
- Two important keys: M(Alt or Meta), C(Ctrl)
- Emacs cheatsheet:

https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf

GNU Emacs Reference Card

(for version 20)

Starting Emacs

To enter GNU Emacs 20, just type its name: `emacs`

To read in a file to edit, see Files, below.

Leaving Emacs

suspend Emacs (or iconify it under X)	`C-z`
exit Emacs permanently	`C-x C-c`

Files

read a file into Emacs	`C-x C-f`
save a file back to disk	`C-x C-s`
save all files	`C-x s`
insert contents of another file into this buffer	`C-x i`
replace this file with the file you really want	`C-x C-v`
write buffer to a specified file	`C-x C-w`
version control checkin/checkout	`C-x C-q`

Getting Help

The help system is simple. Type `C-h` (or `F1`) and follow the directions. If you are a first-time user, type `C-h t` for a tutorial.

remove help window	`C-x 1`
scroll help window	`C-M-v`
apropos: show commands matching a string	`C-h a`
show the function a *key* runs	`C-h c`
describe a function	`C-h f`
get mode-specific information	`C-h m`

Error Recovery

abort partially typed or executing command	`C-g`
recover a file lost by a system crash	`M-x recover-file`
undo an unwanted change	`C-x u` or `C-_`
restore a buffer to its original contents	`M-x revert-buffer`
redraw garbaged screen	`C-l`

Incremental Search

search forward	`C-s`
search backward	`C-r`
regular expression search	`C-M-s`
reverse regular expression search	`C-M-r`
select previous search string	`M-p`
select next later search string	`M-n`
exit incremental search	`RET`
undo effect of last character	`DEL`
abort current search	`C-g`

Use `C-s` or `C-r` again to repeat the search in either direction. If Emacs is still searching, `C-g` cancels only the part not done.

Motion

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 (or end)	`C-a`	`C-e`
sentence	`M-a`	`M-e`
paragraph	`M-{`	`M-}`
page	`C-x [`	`C-x ]`
sexp	`C-M-b`	`C-M-f`
function	`C-M-a`	`C-M-e`
go to buffer beginning (or end)	`M-<`	`M->`

scroll to next screen	`C-v`
scroll to previous screen	`M-v`
scroll left	`C-x <`
scroll right	`C-x >`
scroll current line to center of screen	`C-u C-l`

Killing and Deleting

entity to kill	backward	forward
character (delete, not kill)	`DEL`	`C-d`
word	`M-DEL`	`M-d`
line (to end of)	`M-0 C-k`	`C-k`
sentence	`C-x DEL`	`M-k`
sexp	`M-- C-M-k`	`C-M-k`

kill region	`C-w`
copy region to kill ring	`M-w`
kill through next occurrence of *char*	`M-z` *char*
yank back last thing killed	`C-y`
replace last yank with previous kill	`M-y`

Marking

set mark here	`C-@` or `C-SPC`
exchange point and mark	`C-x C-x`
set mark *arg* words away	`M-@`
mark paragraph	`M-h`
mark page	`C-x C-p`
mark sexp	`C-M-@`
mark function	`C-M-h`
mark entire buffer	`C-x h`

Query Replace

interactively replace a text string	`M-%`
using regular expressions	`M-x query-replace-regexp`

Valid responses in query replace mode are

replace this one, go on to next	`SPC`
replace this one, don't move	`,`
skip to next without replacing	`DEL`
replace all remaining matches	`!`
back up to the previous match	`^`
exit query replace	`RET`
enter recursive edit (`C-M-c` to exit)	`C-r`

Multiple Windows

When two commands are shown, the second is for "other frame."

delete all other windows	`C-x 1`	
split window, above and below	`C-x 2`	`C-x 5 2`
delete this window	`C-x 0`	`C-x 5 0`
split window, side by side	`C-x 3`	
scroll other window	`C-M-v`	
switch cursor to another window	`C-x o`	`C-x 5 o`
select buffer in other window	`C-x 4 b`	`C-x 5 b`
display buffer in other window	`C-x 4 C-o`	`C-x 5 C-o`
find file in other window	`C-x 4 f`	`C-x 5 f`
find file read-only in other window	`C-x 4 r`	`C-x 5 r`
run Dired in other window	`C-x 4 d`	`C-x 5 d`
find tag in other window	`C-x 4 .`	`C-x 5 .`
grow window taller	`C-x ^`	
shrink window narrower	`C-x {`	
grow window wider	`C-x }`	

Formatting

indent current line (mode dependent)	`TAB`
indent region (mode dependent)	`C-M-\`
indent sexp (mode-dependent)	`C-M-q`
indent region rigidly *arg* columns	`C-x TAB`
insert newline after point	`C-o`
move rest of line vertically down	`C-M-o`
delete blank lines around point	`C-x C-o`
join line with previous (with arg, next)	`M-^`
delete all white space around point	`M-\`
put exactly one space at point	`M-SPC`
fill paragraph	`M-q`
set fill column	`C-x f`
set prefix each line starts with	`C-x .`
set face	`M-g`

Case Change

uppercase word	`M-u`
lowercase word	`M-l`
capitalize word	`M-c`
uppercase region	`C-x C-u`
lowercase region	`C-x C-l`

The Minibuffer

The following keys are defined in the minibuffer.

complete as much as possible	`TAB`
complete up to one word	`SPC`
complete and execute	`RET`
show possible completions	`?`
fetch previous minibuffer input	`M-p`
fetch later minibuffer input or default	`M-n`
regexp search backward through history	`M-r`
regexp search forward through history	`M-s`
abort command	`C-g`

Type `C-x ESC ESC` to edit and repeat the last command that used the minibuffer. Type `F10` to activate the menu bar using the minibuffer.

GNU Emacs Reference Card

Buffers

select another buffer	C-x b
list all buffers	C-x C-b
kill a buffer	C-x k

Transposing

transpose characters	C-t
transpose words	M-t
transpose lines	C-x C-t
transpose sexps	C-M-t

Spelling Check

check spelling of current word	M-$
check spelling of all words in region	M-x ispell-region
check spelling of entire buffer	M-x ispell-buffer

Tags

find a tag (a definition)	M-.
find next occurrence of tag	C-u M-.
specify a new tags file	M-x visit-tags-table
regexp search on all files in tags table	M-x tags-search
run query-replace on all the files	M-x tags-query-replace
continue last tags search or query replace	M-,

Shells

execute a shell command	M-!
run a shell command on the region	M-|
filter region through a shell command	C-u M-|
start a shell in window *shell*	M-x shell

Rectangles

copy rectangle to register	C-x r r
kill rectangle	C-x r k
yank rectangle	C-x r y
open rectangle, shifting text right	C-x r o
blank out rectangle	C-x r c
prefix each line with a string	C-x r t

Abbrevs

add global abbrev	C-x a g
add mode-local abbrev	C-x a l
add global expansion for this abbrev	C-x a i g
add mode-local expansion for this abbrev	C-x a i l
explicitly expand abbrev	C-x a e
expand previous word dynamically	M-/

Regular Expressions

any single character except a newline	. (dot)
zero or more repeats	*
one or more repeats	+
zero or one repeat	?
quote regular expression special character c	\c
alternative ("or")	|
grouping	\(... \)
same text as nth group	\n
at word break	\b
not at word break	\B

entity	match start	match end
line	^	$
word	\<	\>
buffer	\`	\'
class of characters	**match these**	**match others**
explicit set	[...]	[^ ...]
word-syntax character	\w	\W
character with syntax c	\sc	\Sc

International Character Sets

specify principal language	M-x set-language-environment
show all input methods	M-x list-input-methods
enable or disable input method	C-\
set coding system for next command	C-x RET c
show all coding systems	M-x list-coding-systems
choose preferred coding system	M-x prefer-coding-system

Info

enter the Info documentation reader	C-h i
find specified function or variable in Info	C-h C-i

Moving within a node:

scroll forward	SPC
scroll reverse	DEL
beginning of node	. (dot)

Moving between nodes:

next node	n
previous node	p
move up	u
select menu item by name	m
select nth menu item by number (1–9)	n
follow cross reference (return with l)	f
return to last node you saw	l
return to directory node	d
go to any node by name	g

Other:

run Info tutorial	h
quit Info	q
search nodes for regexp	M-s

Registers

save region in register	C-x r s
insert register contents into buffer	C-x r i
save value of point in register	C-x r SPC
jump to point saved in register	C-x r j

Keyboard Macros

start defining a keyboard macro	C-x (
end keyboard macro definition	C-x)
execute last defined keyboard macro	C-x e
append to last keyboard macro	C-u C-x (
name last keyboard macro	M-x name-last-kbd-macro
insert Lisp definition in buffer	M-x insert-kbd-macro

Commands Dealing with Emacs Lisp

eval sexp before point	C-x C-e
eval current defun	C-M-x
eval region	M-x eval-region
read and eval minibuffer	M-:
load from standard system directory	M-x load-library

Simple Customization

customize variables and faces	M-x customize

Making global key bindings in Emacs Lisp (examples):

```
(global-set-key "\C-cg" 'goto-line)
(global-set-key "\M-#" 'query-replace-regexp)
```

Writing Commands

```
(defun command-name (args)
  "documentation" (interactive "template")
  body)
```

An example:

```
(defun this-line-to-top-of-window (line)
  "Reposition line point is on to top of window.
With ARG, put point on line ARG."
  (interactive "P")
  (recenter (if (null line)
                0
              (prefix-numeric-value line))))
```

The `interactive` spec says how to read arguments interactively. Type `C-h f interactive` for more details.

For copies of the GNU Emacs manual, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111 1307 USA

Emacs basics

- C-t : tutorial
- C-x C-c: exit
- Emacs filename
- C-g: cancel partially typed or accidental command
- C-x u: undo last change

Emacs on files

- Navigating with file
  - Move up/down/left/right: C-p, C-n, C-b, C-f (or arrow keys)
  - Move to the beginning/end of a line: C-a, C-e
  - Move to the first/last line of the text: M-<, M->
- Search and replace file
  - C-s: search forward
  - C-r: search backward
  - M-%: replace (usage: M-% [source] Enter [dest])
- Erasing a line
  - C-k: erase from current cursor to end of line
- Copy and paste in a file
  - Begin: C-@ (press Ctrl+Shift+2)
  - Use the <up> and <down> buttons to select the contents
  - End: C-w (cut), M-w(copy), C-y (paste)
  - Undo command: C-u

Extended commands

- Use C-x plus other combined button
  - New file: C-x C-f
  - Quit Emacs: C-x C-c
  - f a file is modified, it will ask you whether to save the file and whether to leave now. (input y, yes)

Compile with Emacs

- Visit *scratch* buffer

  C-x b

- Compiling ***C*** code with emacs

  M-x compile

- Running Lisp code

  M-x emacs-lisp-mode

- C-x C-e : Evaluate expression up to point

http://www.emacswiki.org/emacs/EvaluatingExpressions

Format of submission

- For lab questions(ans1.txt)
  - Answer 15 questions using natural language
  - List all the commands used to solve the problem
  - Give some explanations about your choice of commands
  - No need for keystrokes in this file
  - Will be graded manually

Format of submission

- For exercises (key1.txt)
  - Record keystroke of each exercise separately
  - Don’t forget commands to enter/leave emacs
  - The keystrokes for different exercise should be recorded separately, each keystroke for one line

```
Exersise 1.1
1. e m a c s SP assign1.html Enter
2. C-s T
3. C-b
.
.
.
11. C-x C-c

Exersise 1.2
1. e m a c s SP assign2.html Enter
.
.
.
5. ..... Backspace Backspace
```

Some Tips

- Just use simple commands, this is a warm-up task. Take it easy.
- If you are not sure about complex commands, just use the combination of simple ones

  e.g. <left> <left> …… <left> = C-a
- If you don’t know the exact answers, just write down what you thought and what you have tried

Submit your homework

- Use scp to copy files from server to your local machine
  - Google how to use
- Submit your two files in ccle
- Lab file must be named as ans1.txt

  Homework file must be named as key1.txt

Week1 Check List

- Multiuser and multiprocess OS
- GUI basics
- CLI basics
- Unix file system layout
- Unix permission
- Basic commands
- Documentations and man pages
- Emacs basics

Shell Scripting

Week 2

Administration

- My office hour:

  Thursdays 09:30–11:30 @BH 2432

- Assignment 2 is due Saturday 4/15 11:55pm on CCLE
  - Probably the most difficult assignment
  - Start early
- http://web.cs.ucla.edu/classes/spring17/cs35L/assign.html
- Lab part: buildwords, lab2.log

  Homework part: sameln

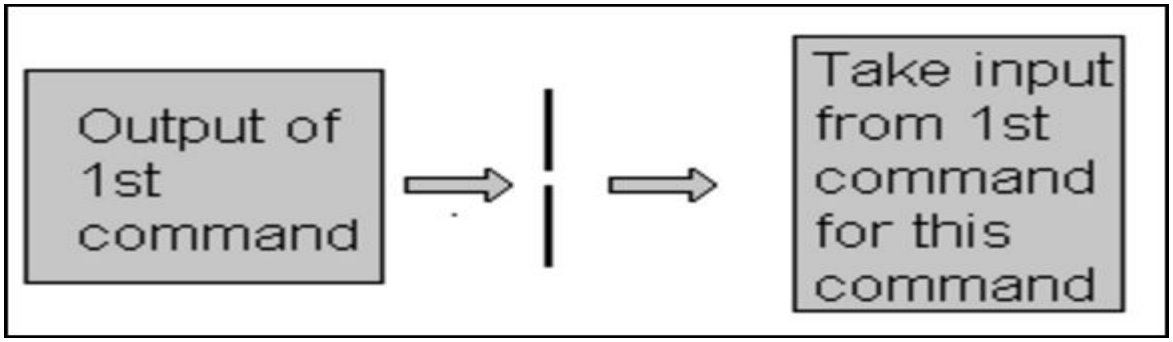
The Basics: Redirection

- > *file*: (over)write stdout to a file
- >> *file*: append stdout to a file
- < *file*: use contents of a file as stdin

Pipe

Command 1 | command 2

- **connect the output of command1 to the input of command 2 without any temporary file.**



$ **who**

```
george          pts/2  Dec 31 16:39 (valley-forge.example.com)
betsy     pts/3  Dec 27 11:07 (flags-r-us.example.com)
benjamin  dtlocal        Dec 27 17:55 (kites.example.com)
jhancock  pts/5  Dec 27 17:55 (:32)
Camus           pts/6  Dec 31 16:22
tolstoy         pts/14         Jan 2 06:42
```

$ **who | wc -l** *Count users*

6

Text Processing Tools

- sort: sorts text alphabetically
- wc: count the number of lines, words, and bytes
- head: extract top of files (default 10 lines)
- tail: extracts bottom of files (default 10 lines)

ex: head -n 20 lab2.txt

Q: how to get line 10 to line 20 of a file?

The tr command

- Abbreviated as translate or transliterate
- Usage

  tr [options] [set1] [set2]

- Function: replace the elements in set1 with corresponding elements from set2

  Example: tr [a-z] [A-Z]

-- replace all lowercase letters from input with uppercase letters

-- input can be stdin, file, pipe

-- tr [:lower:] [:upper:]

Options in tr command

tr [options] [set1] [set2]

- -C , -- complement

  - use the complement of set 1

- -d, -- delete

  - delete characters in SET1, do not translate

- -s, -- squeeze-repeats

  - replace each input sequence of a repeated character that is listed in SET1 with a single occurrence of that character

Question: tr -c 'A-Za-z' '[\n*]’

Examples of tr command

- Usage: as a part of pipeline
  - e.g. cat assign2.html | tr -cs 'A-Za-z' '[\n*]' > pre
- Eliminate everything except alphabet characters, also duplicate words
  - tr -cs 'A-Za-z' '[\n*]'
- Transform all upper cases characters to lower cases
  - tr '[:upper:]' '[:lower:]'
- Delete all left-over blanks
  - tr -d '[:blank:]'

sed command

sed [options] … {script-only-if-no-other-script} [input-file] ...

- Stream editor for filtering and transforming text
- Modify input as specified by the commands
- Use sed to search patterns and replace
  - sed ‘s/regexp/replacement/”
  - Ex: sed 's/h.llo/world/'

```
sed 's/\*/\+/g'
sed 's/ /\n/g'
    sed 's/:.*//'
```

Note: g means global, operate on the whole line

sed command

sed [options] … {script-only-if-no-other-script} [input-file] ...

- Use sed to delete lines with certain number or pattern

- sed '3d'

  (delete the third line)

- sed '3,$d'

  (delete from the third line to the end)

- sed '/pattern/d'

  (delete lines with certain pattern)

- sed '/pattern1/ , /pattern2/d'

  (delete a range of lines, pattern1 turns on the deletion, pattern 2 turns off the deletion)

Compare difference between files

- diff
  - usage: diff [option] [file1] [file2]
  - function: compare files line by line
- comm
  - usage: comm [option] [file1] [file2]
  - function: compare sorted files line by line

Laboratory -- Spell-checking Hawaiian

- Finish the script *buildword*:
  - Basic Structure: Using a pipeline of tr and sed commands
  - Input: read html from stdin
  - Output: a sorted list of unique words
  - Usage: cat foo.html bar.html | ./buildwords
  - Preprocess
    - Delete whatever before/after the html <table> tag
    - Eliminate html tags, extract words

Laboratory -- Spell-checking Hawaiian

- First clean the web pages:
  - Eliminate characters expect a-z and A-Z
  - Eliminate characters out of Hawaiian alphabet
  - Transform upper cases to lower cases
- Use commands: tr and sed
- Try basic regular expressions

Laboratory -- Spell-checking Hawaiian

- Finish the script *buildword* (continue):
  - Change upper case characters to lower case
  - Treat ` as ’
  - Remove any misspelled Haiwaiin language
  - Hints: don’t leave unnecessary information behind (e.g. duplication, empty lines, spaces, html tags)

grep command

grep [options] pattern [file...]

- Global regular expression print
- Process text line by line and print lines matching a specified pattern
- Options
- f: obtain patterns from file, one per line
- i: ignore cases in both patterns and input files
- Similar commands
- grep uses basic regular expression (BRE)
- egrep (-E) uses extend regular expression (ERE)
- fgrep (-F) uses fixed strings instead of regular expressions

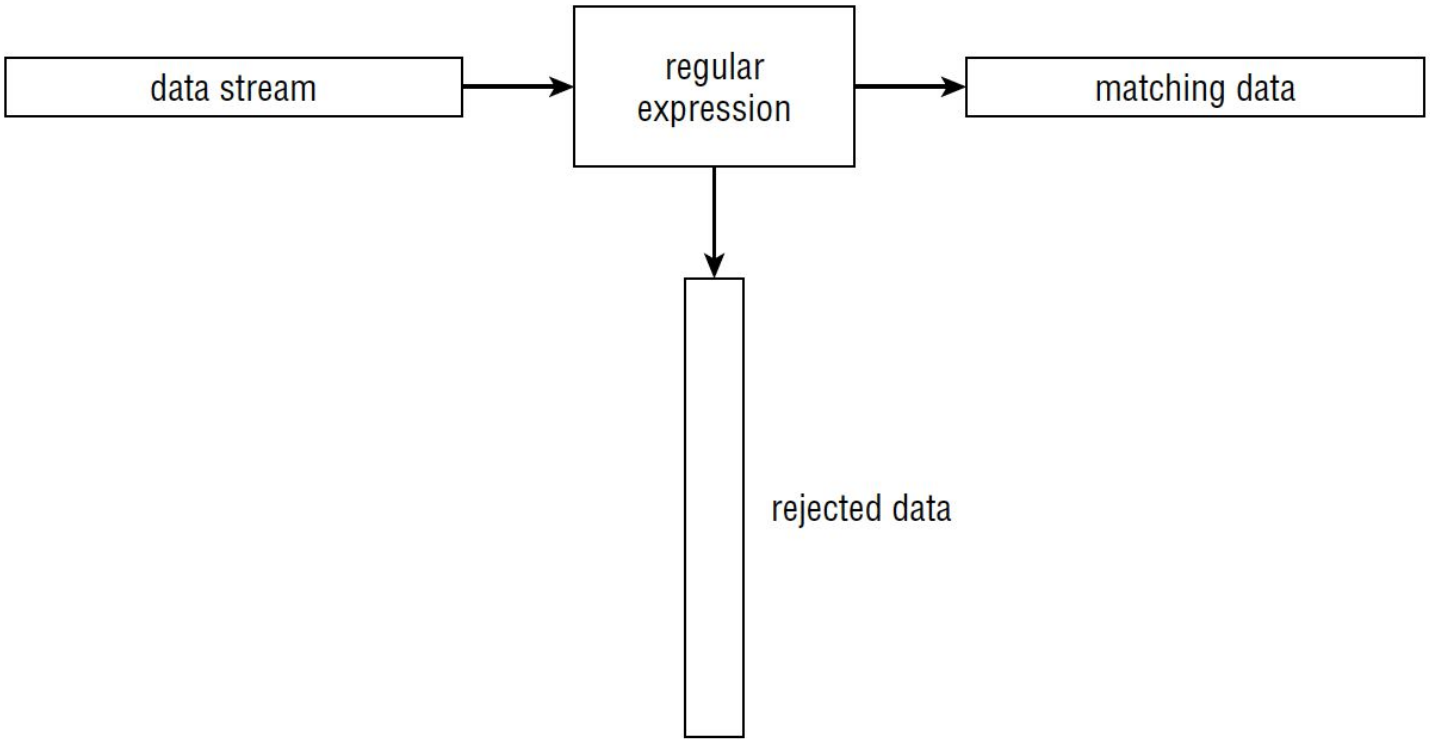
Simple grep

```
$ who                          Who is logged on
tolstoy tty1 Feb 26 10:53
tolstoy pts/0 Feb 29 10:59
tolstoy pts/1 Feb 29 10:59
tolstoy pts/2 Feb 29 11:00
tolstoy pts/3 Feb 29 11:00
tolstoy pts/4 Feb 29 11:00
austen pts/5 Feb 29 15:39 (mansfield-park.example.com)
austen pts/6 Feb 29 15:39 (mansfield-park.example.com)

$ who | grep -F austen          Where is austen logged on?
austen pts/5 Feb 29 15:39 (mansfield-park.example.com)
austen pts/6 Feb 29 15:39 (mansfield-park.example.com)
```

Regular Expressions

- Notation that represents a text pattern
  - ex: starts with the letter a
- Match/filter data against a regular expression



Regular Expressions

- Different applications use different types of regular expression
  - Programming languages (i.e. Java, Perl, Python)
  - Linux utilities (i.e. sed,grep)
  - Mainstream applications (i.e. MySQL)
- Regular Expression Engine
  - Interprets regular patterns and use patterns to match text
  - The POSIX Basic Regular Expression (BRE) engine
  - The POSIX Extended Regular Expression (ERE) engine

Special characters in Regular expression

- Quantification (the number of previous occurrences)
  - ? (0 or 1)
  - * (0 or more)
  - + (1 or more)
  - {} (specified number)
- Alternation
  - [] (any character in the range)
  - | (one case or another)
- Anchors
  - ^ (beginning of a line)
  - $ (end of a line)
- Group
  - ()

Regular expressions

Character	BRE / ERE	Meaning in a pattern
\	Both	Usually, turn off the special meaning of the following character. Occasionally, enable a special meaning for the following character, such as for \(...\) and \{...\}.
.	Both	Match any single character except NUL. Individual programs may also disallow matching newline.
*	Both	Match any number (or none) of the single character that immediately precedes it. For EREs, the preceding character can instead be a regular expression. For example, since . (dot) means any character, .* means "match any number of any character." For BREs, * is not special if it's the first character of a regular expression.
^	Both	Match the following regular expression at the beginning of the line or string. BRE: special only at the beginning of a regular expression. ERE: special everywhere.

Regular Expressions (cont'd)

$	Both	Match the preceding regular expression at the end of the line or string. BRE: special only at the end of a regular expression. ERE: special everywhere.
[...]	Both	Termed a bracket expression, this matches any one of the enclosed characters. A hyphen (-) indicates a range of consecutive characters. (Caution: ranges are locale-sensitive, and thus not portable.) A circumflex (^) as the first character in the brackets reverses the sense: it matches any one character not in the list. A hyphen or close bracket (]) as the first character is treated as a member of the list. All other metacharacters are treated as members of the list (i.e., literally). Bracket expressions may contain collating symbols, equivalence classes, and character classes (described shortly).
\{*n*,*m*\}	BRE	Termed an *interval expression*, this matches a range of occurrences of the single character that immediately precedes it. \{*n*\} matches exactly n occurrences, \{*n*,\} matches at least n occurrences, and \{*n*,*m*\} matches any number of occurrences between n and m. n and m must be between 0 and RE_DUP_MAX (minimum value: 255), inclusive.
\(\)	BRE	Save the pattern enclosed between \(and \) in a special *holding space*. Up to nine subpatterns can be saved on a single pattern. The text matched by the subpatterns can be reused later in the same pattern, by the escape sequences \1 to \9. For example, **\(ab\).*\1** matches two occurrences of ab, with any number of characters in between.

Regular Expressions (cont'd)

\n	BRE	Replay the nth subpattern enclosed in \(and \) into the pattern at this point. n is a number from 1 to 9, with 1 starting on the left.
{n,m}	ERE	Just like the BRE \{n,m\} earlier, but without the backslashes in front of the braces.
+	ERE	Match one or more instances of the preceding regular expression.
?	ERE	Match zero or one instances of the preceding regular expression.
|	ERE	Match the regular expression specified before or after.
( )	ERE	Apply a match to the enclosed group of regular expressions.

Examples

Expression	Matches
tolstoy	The seven letters tolstoy, anywhere on a line
^tolstoy	The seven letters tolstoy, at the beginning of a line
tolstoy$	The seven letters tolstoy, at the end of a line
^tolstoy$	A line containing exactly the seven letters tolstoy, and nothing else
[Tt]olstoy	Either the seven letters Tolstoy, or the seven letters tolstoy, anywhere on a line
tol.toy	The three letters tol, any character, and the three letters toy, anywhere on a line
tol.*toy	The three letters tol, any sequence of zero or more characters, and the three letters toy, anywhere on a line (e.g., toltoy, tolstoy, tolWHOtoy, and so on)

BRE Special Character Classes

Class	Description
`[[:alpha:]]`	Match any alphabetical character, either upper or lower case.
`[[:alnum:]]`	Match any alphanumeric character 0–9, A–Z, or a–z.
`[[:blank:]]`	Match a space or Tab character.
`[[:digit:]]`	Match a numerical digit from 0 through 9.
`[[:lower:]]`	Match any lower-case alphabetical character a–z.
`[[:print:]]`	Match any printable character.
`[[:punct:]]`	Match a punctuation character.
`[[:space:]]`	Match any whitespace character: space, Tab, NL, FF, VT, CR.
`[[:upper:]]`	Match any upper-case alphabetical character A–Z.

Shell Scripting

Week 2

Shell Script

- The ability to enter multiple commands and combine them logically
- Must specify the shell you use in the first line
  - #!/bin/bash
    (# itself can lead comments)
- You can create easiest shell script by listing commands in separate lines

  (shell will process commands in order)

Q: How to write a shell script that can find the logged in user with username “betsy”?

- What commands to use?
- How to write and execute the script?

Example:

```
$ who | grep betsy                    Where is betsy?
betsy pts/3 Dec 27 11:07 (flags-r-us.example.com)
```

Script:

```
#! /bin/sh
# finduser --- see if user named by betsy is logged in
who | grep betsy
```

Run it:

```
$ chmod +x finduser                   Make it executable
$ ./finduser                          Test it: find betsy
betsy pts/3 Dec 27 11:07 (flags-r-us.example.com)
```

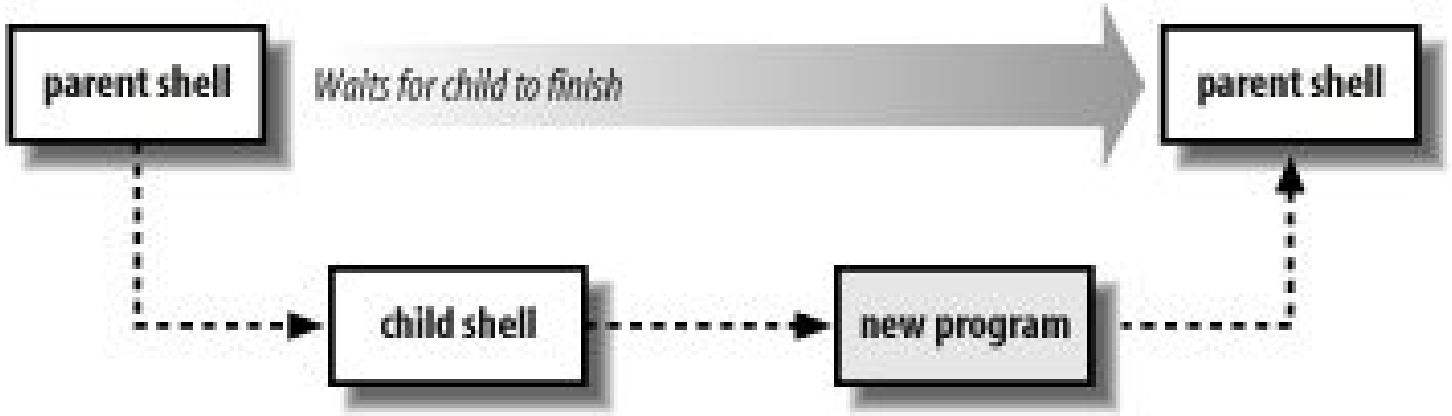
Self-Contained Scripts: The #! First Line

- When the shell runs a program, it asks the kernel to start a new process and run the given program in that process.
- But if there is more than one shell installed on the system, we need a way to tell the kernel which shell to use for a script

  #! /bin/csh –f

  #! /bin/awk –f

  #! /bin/sh
- A shell can initiate a shell different from itself



Variables

- Allows you to temporarily store info and use it later
- Two general types
  - Environment variables
  - User defined variables (UDV)
- Environment variables
  - Created and maintained by Linux itself
  - Track specific system info
  - defined in CAPITAL LETTERS
  - ex: $PATH, $PWD
- User defined variables (UDV)
  - Created and maintained by user
  - defined in lower letters

- When refer a variable value, use dollar sign
  - echo $PWD
- When refer a variable to assign a value to it, do not use dollar sign (no space around =)
  - myvar=helloworld
- export: puts variables into the environment
  - Environment is a list of name-value pairs that is available to every running program
- env: Displays the current environment
- unset: remove variable and functions from the current shell

POSIX Built-in Shell Variables

Variable	Meaning
#	Number of arguments given to current process.
@	Command-line arguments to current process. Inside double quotes, expands to individual arguments.
*	Command-line arguments to current process. Inside double quotes, expands to a single argument.
- (hyphen)	Options given to shell on invocation.
?	Exit status of previous command.
$	Process ID of shell process.
0 (zero)	The name of the shell program.
!	Process ID of last background command. Use this to save process ID numbers for later use with the *wait* command.
ENV	Used only by interactive shells upon invocation; the value of $ENV is parameter-expanded. The result should be a full pathname for a file to be read and executed at startup. This is an XSI requirement.
HOME	Home (login) directory.
IFS	Internal field separator; i.e., the list of characters that act as word separators. Normally set to space, tab, and newline.
LANG	Default name of current locale; overridden by the other LC_* variables.
LC_ALL	Name of current locale; overrides LANG and the other LC_* variables.
LC_COLLATE	Name of current locale for character collation (sorting) purposes.
LC_CTYPE	Name of current locale for character class determination during pattern matching.
LC_MESSAGES	Name of current language for output messages.
LINENO	Line number in script or function of the line that just ran.
NLSPATH	The location of message catalogs for messages in the language given by $LC_MESSAGES (XSI).
PATH	Search path for commands.
PPID	Process ID of parent process.
PS1	Primary command prompt string. Default is "$ ".
PS2	Prompt string for line continuations. Default is "> ".
PS4	Prompt string for execution tracing with set -x. Default is "+ ".
PWD	Current working directory.

- PATH
  - list of directories in which commands are found
  - echo $PATH
  - export $PATH = /usr/local/bin : $PATH

    (Prepend a new dic path in $PATH)
- ?
  - Show exit status of previous command
  - 0 means exit normally
  - Otherwise exit with some errors
- $IFS
  - Define a list of field separators

Linux Exit Status Codes

Code	Description
0	Successful completion of the command
1	General unknown error
2	Misuse of shell command
126	The command can't execute
127	Command not found
128	Invalid exit argument
128+x	Fatal error with Linux signal x
130	Command terminated with Ctl-C
255	Exit status out of range

Quote

- Single quote '
  - Literal meaning of everything within ''
  - echo '$PATH'
- Double quote "
  - Literal meaning of everything except $ \ `
  - echo "the current directory is $PWD"
- The backtick `
  - Execute the command
  - Allow you to assign the output of a shell command to a variable
  - testing `date`

Q: How to write a shell script that can find the logged in user with username “edison”?

- Can I generalize the script to find any user name?

Command Line Parameters

- Allow user to pass data to script before execution
- Positional parameters
  - represent all parameters entered in a command line
- $0: the name of the program

  $1: the first parameter

  ${10}: the 10th parameter

Example:

```
$ who | grep betsy                    Where is betsy?
betsy pts/3 Dec 27 11:07 (flags-r-us.example.com)
```

Script:

```
#! /bin/sh
# finduser --- see if user named by first argument is logged in
who | grep $1
```

Run it:

```
$ chmod +x finduser                   Make it executable
$ ./finduser betsy                    Test it: find betsy
betsy pts/3 Dec 27 11:07 (flags-r-us.example.com)
$ ./finduser benjamin                 Now look for Ben
 benjamin dtlocal Dec 27 17:55 (kites.example.com)
```

Structured commands

- Alter the flow of operations based conditions
- If statement
- For statement
- While loops
- Case statement
- Break statement
- Continue statement

IF-THEN Statement

- ```
  if command
  then
       commands
  fi
  ```
- If the exit status of command is zero (complete successfully), the command listed under then *then* section are executed
- ```
  #!/bin/bash
  # testing the if statement
  if date
  then
     echo "it worked"
  fi
  ```

More IF Statement

```
if command
then
    commands
else
    commands
fi
```

```
if command1
then
     command set 1
elif command2
then
    command set 2
elif command3
then
    command set 3
elif command4
then
    command set 4
fi
```

Test command

- The ability to evaluate any condition other than the exit code of a status
  (i.e. evaluate true/false)
-
```
if test condition
then
    commands
fi
```

```
if [ condition ]
then
    commands
fi
```

- If the condition listed in the test command is true, the test command exits with 0

Test command

- Three classes of conditions
  - Numeric comparisons
  - String comparisons
  - File comparisons

```
if test condition
then
    commands
fi
```

```
if [ condition ]
then
    commands
fi
```

- Numeric comparisons
- Evaluate both numbers and variables
- Ex: $var -eq 1
  $var1 -ge $var2

The test Numeric Comparisons

Comparison	Description
n1 -eq *n2*	Check if *n1* is equal to *n2*.
n1 -ge *n2*	Check if *n1* is greater than or equal to *n2*.
n1 -gt *n2*	Check if *n1* is greater than *n2*.
n1 -le *n2*	Check if *n1* is less than or equal to *n2*.
n1 -lt *n2*	Check if *n1* is less than *n2*.
n1 -ne *n2*	Check if *n1* is not equal to *n2*.

- String comparisons
- The greater-than and less-than symbols must be escaped (otherwise will be interpreted as redirection)
- The greater-than and less-than order is not the same as sort (ASCII vs. locale language)
- Ex: $USER = $testuser

The test Command String Comparisons

Comparison	Description
str1 = *str2*	Check if *str1* is the same as string *str2*.
str1 != *str2*	Check if *str1* is not the same as *str2*.
str1 < *str2*	Check if *str1* is less than *str2*.
str1 > *str2*	Check if *str1* is greater than *str2*.
-n *str1*	Check if *str1* has a length greater than zero.
-z *str1*	Check if *str1* has a length of zero.

- File comparisons
  - Test the status of files and directories in linux file system

The test Command File Comparisons

Comparison	Description
`-d file`	Check if *file* exists and is a directory.
`-e file`	Checks if *file* exists.
`-f file`	Checks if *file* exists and is a file.
`-r file`	Checks if *file* exists and is readable.
`-s file`	Checks if *file* exists and is not empty.
`-w file`	Checks if *file* exists and is writable.
`-x file`	Checks if *file* exists and is executable.
`-O file`	Checks if *file* exists and is owned by the current user.
`-G file`	Checks if *file* exists and the default group is the same as the current user.
file1 `-nt` *file2*	Checks if *file1* is newer than *file2*.
file1 `-ot` *file2*	Checks if *file1* is older than *file2*.

FOR Statement

```
for var in list
do
   commands
done
```

```
for (( i=1; i <= 10; i++ ))
do
   echo "The next number is $i"
done
```

- For each value in the list, do a set of operations on it
- example

```
for test in Alabama Alaska Arizona
do
   echo The next state is $test
done
```

- Q: how does computer know how to split the list?

Q: how does computer know how to split the list?

- $IFS
  - Internal field separator
  - Define a list of characters the bash shell uses as field separators
  - Default values: space, tab, newline
  - Can change value of $IFS to split list in different ways
  - Better store original values and restore later

```
IFS.OLD=$IFS
IFS=$'\n'
<use the new IFS value in code>
IFS=$IFS.OLD
```

Other loop statements

```
while test command
do
  other commands
done
```

```
until test commands
do
    other commands
done
```

- In while loop, commands in the loop are executed as long as the exit code of test command is 0
- In until loop, as long as the exit status of the test command is non-zero, commands listed in the loop are executed

Homework: find duplicate files

- Input argument: the path of directory
- Usage: ./sameln [directory name]
- Output: a list of regular files immediately under the given directory which have duplicates
- First line: #!/bin/bash

Homework: find duplicate files

- Some tips
  - Only consider files immediately under given directory
    (hints: find -maxdepth 1 -type f)
  - For duplicates, keep the one whose name is lexicographically first, replace other files with hard links to the first one
    (hints: use ln command)
  - Don't forget hidden files that begin with . !
    (hints: ls -a [directory] | grep '^\.')
  - Ignore non-regular and not readable files
  - File names may contain special characters (e.g. space, *, -)

Scripting Languages VS Compiled Languages

- Compiled Languages
  - Programs are translated from human-readable code to machine-readable code by compiler
  - Efficient
  - Ex: C/C++, Java
- Scripting languages
  - rely on source-code all the time
  - Interpreter reads program, translates it into internal form, and execute programs on the fly
  - Inefficient (translation on the fly)
  - Ex: Python, Ruby, PHP, Perl

Modify and Rewrite Programs

Week 3

Scripting Languages VS Compiled Languages

- Compiled Languages
  - Programs are translated from human-readable code to machine-readable code by compiler
  - Efficient
  - Ex: C/C++, Java
- Scripting languages
  - rely on source-code all the time
  - Interpreter reads program, translates it into internal form, and execute programs on the fly
  - Inefficient (translation on the fly)
  - Ex: Python, Ruby, PHP, Perl

How to Install Software

- Windows
  - Installshield
  - Microsoft/Windows Installer
- OS X
  - Drag and drop from .dmg mount -> Applications folder
- Linux
  - rpm(Redhat Package Management)
    - RedHat Linux (.rpm)
  - apt-get(Advanced Package Tool)
    - Debian Linux, Ubuntu Linux (.deb)
  - **Good old build process**
    - **configure, make, make install**

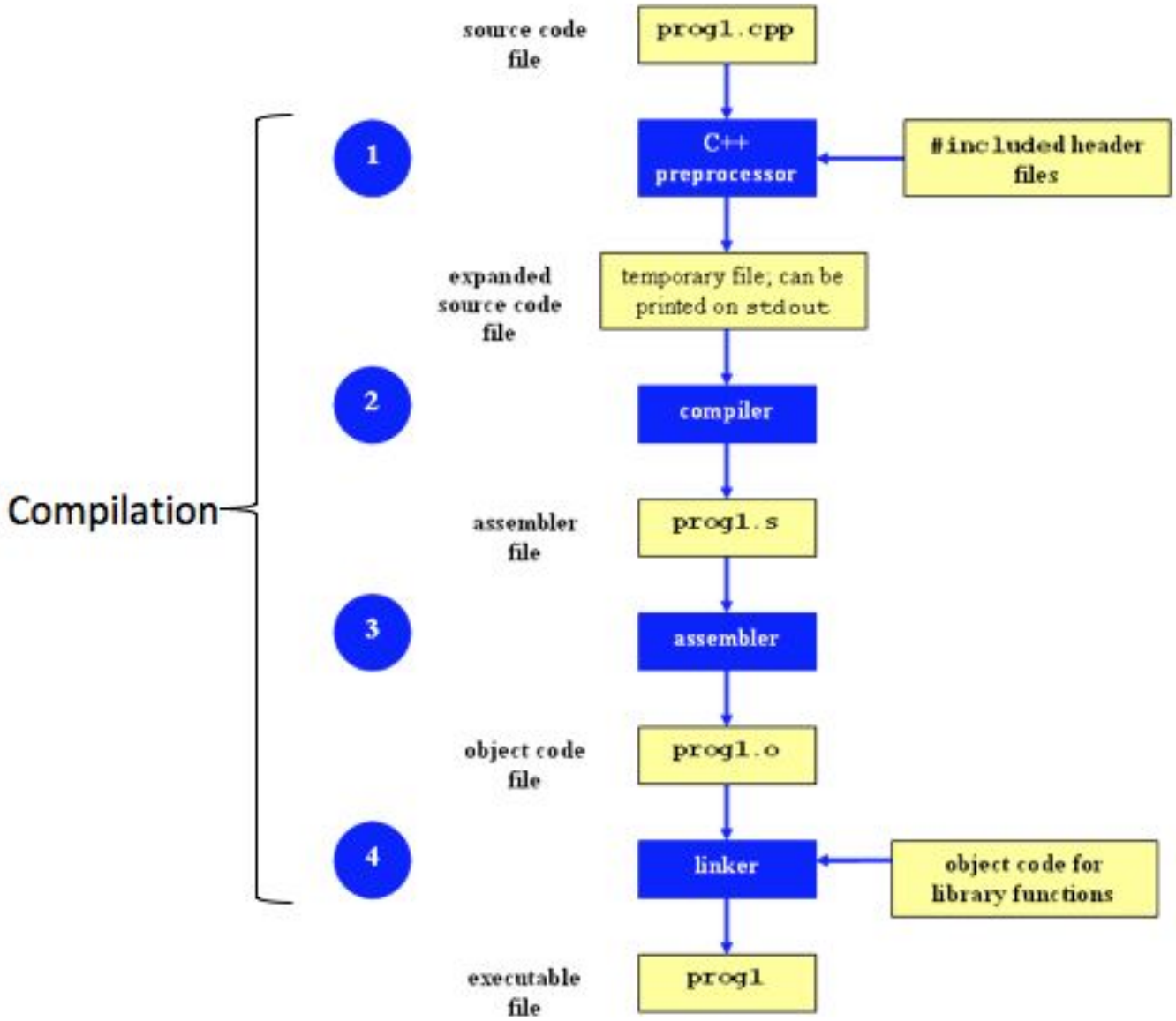
Decompressing Files

- Generally, you receive Linux software in the tarball format (.tgz) or (.gz)

Decompress file in current directory:

- $ tar –xzvf filename.tar.gz
  - Option –x: --extract
  - Option –z: --gzip
  - Option –v: --verbose
  - Option –f: --file

Compilation Process



Command-Line Compilation

- shop.cpp
  - #includes shoppingList.h and item.h
- shoppingList.cpp
  - #includes shoppingList.h
- item.cpp
  - #includes item.h
- How to compile?
  - **g++ -Wall shoppingList.cpp item.cpp shop.cpp –o shop**

What if…

- **We change one of the header or source files?**
  - Rerun command to generate new executable
- **We only made a small change to item.cpp?**
  - not efficient to recompile shoppinglist.cpp and shop.cpp
  - Solution: avoid waste by producing a separate object code file for each source file
    - g++ -Wall –c item.cpp… (for each source file)
    - g++ item.o shoppingList.o shop.o –o shop (combine)
    - Less work for compiler, saves time but more commands

What if…

- **We change item.h?**
  - Need to recompile every source file that includes it & every source file that includes a header that includes it. Here: item.cpp and shop.cpp
  - Difficult to keep track of files when project is large
    - Windows 7 ~40 million lines of code
    - Google ~2 billion lines of code

=> Make

Make

make [OPTION]... [TARGET]...

- GNU utilities to maintain groups of program
- Automatically determine which part of large program needs to be recompiled
  - Make update a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist
- Efficient compilation
- Take a Makefile to specify all target and prerequisite

Makefile Example

```
# Makefile - A Basic Example
all : shop  #usually first
shop : item.o shoppingList.o shop.o
        g++ -g -Wall -o shop item.o shoppingList.o shop.o
item.o : item.cpp item.h
        g++ -g -Wall -c item.cpp
shoppingList.o : shoppingList.cpp shoppingList.h
        g++ -g -Wall -c shoppingList.cpp
shop.o : shop.cpp item.h shoppingList.h
        g++ -g -Wall -c shop.cpp
clean :
        rm -f item.o shoppingList.o shop.o shop
```

Rule

Comments

Targets

Prerequisites

Dependency Line

Commands

Build Process

- **configure**
  - Script that checks details about the machine before installation
    - Dependency between packages
  - Creates 'Makefile'
- **make**
  - Requires 'Makefile' to run
  - Compiles all the program code and creates executables in current temporary directory
- **make install**
  - make utility searches for a label named install within the Makefile, and executes only that section of it
  - executables are copied into the final directories (system directories)

```
./configure
make
make install
```

Lab 3

- Coreutils 7.6 has a problem
  - Different users see different date formats
  - $ ls –l /bin/bash
    - -rwxr-xr-x 1 root root 729040 **2009-03-02 06:22** /bin/bash
    - -rwxr-xr-x 1 root root 729040 **Mar 2 2009** /bin/bash
- Why?
  - Different locales
- Want the traditional Unix format for all users
- Fix the ls program

Getting Set Up (Step 1)

- Download coreutils-7.6 to your home directory
  - Use ‘wget’
- Untar and Unzip it
  - tar –xzvf coreutils-7.6.tar.gz
- Make a directory ~/coreutilsInstall in your home directory (this is where you’ll be installing coreutils)
  - mkdir coreutilsInstall

Building coreutils (Step 2)

- Go into coreutils-7.6 directory. This is what you just unzipped.
- Read the INSTALL file on how to configure "make", especially **--prefix** flag
- Run the configure script using the prefix flag so that when everything is done, coreutils will be installed in the directory ~/coreutilsInstall
- Compile it: make
- Install it: make install (won't work on Linux server without proper prefix!)
  - Why?

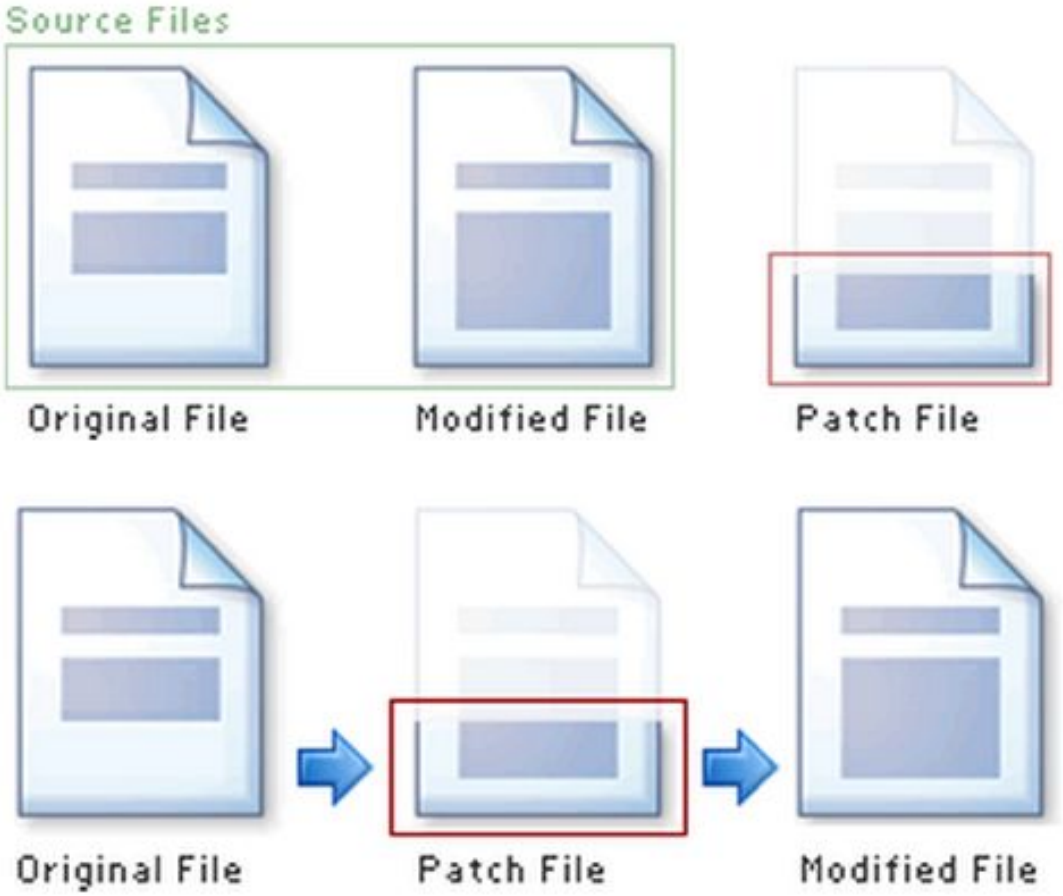
Reproduce Bug (Step 3)

- Reproduce the bug by running the version of 'ls' in coreutils 7.6
- If you just type $ ls at CLI it won't run 'ls' in coreutils 7.6
  - Why? Shell looks for /bin/ls
  - To use coreutils 7.6: $ ./ls
    - This manually runs the executable in this directory

Patching

- A patch is a piece of software designed to fix problems with or update a computer program
- It's a diff file that includes the changes made to a file
- A person who has the original (buggy) file can use the patch command with the diff file to add the changes to their original file

Applying a Patch



diff Unified Format

- diff –u original_file modified_file

- --- path/to/original_file
- +++ path/to/modified_file

- @@ -l,s +l,s @@
  - @@: beginning of a hunk
  - l: beginning line number
  - s: number of lines the change hunk applies to for each file
  - A line with a:
    - - sign was deleted from the original
    - + sign was added to the original
    - stayed the same

Applying the Patch

- Download the patch

```
Index: src/df.c
===================================================================
RCS file: /cvsroot/coreutils/coreutils/src/df.c,v
retrieving revision 1.168
diff -p -d -U6 -r1.168 df.c
--- src/df.c   16 Aug 2005 20:33:40 -0000 1.168
+++ src/df.c   12 Oct 2005 06:10:18 -0000
@@ -297,12 +297,14 @@ show_dev (char const *disk, char const *
     but statfs doesn't do that on most systems. */
    if (!stat_file)
     stat_file = mount_point ? mount_point : disk;
    if (get_fs_usage (stat_file, disk, &fsu))
        {
+    if(errno == EACCES && !show_all_fs && !show_listed_fs)
+    return; /* Ignore mount points we can't access */
     error (0, errno, "%s", quote (stat_file));
     exit_status = EXIT_FAILURE;
     return;
        }
     if (fsu.fsu_blocks == 0 && !show_all_fs && !show_listed_fs)
```

Patching and Building (Steps 4 & 5)

- cd coreutils-7.6
- vim or emacs patch_file: copy and paste the patch content
- `patch` –**pnum** < patch_file
  - '`man patch`' to find out what **pnum** does and how to use it
- `cd` into the coreutils-7.6 directory and type make to rebuild patched ls.c.
  - Don't install!!

Testing Fix (Step 6)

- Test the following:
  - Modified ls works
  - Installed unmodified ls does NOT work
- Test on:
  - 1) a file that has been recently modified
    - Make a change to an existing file or create a new file
  - 2) a file that is at least a year old
    - touch –t 201401210959.30 *test_file*
- Answer Q1 and Q2

Modify and Rewrite Programs

Week 3

Make

make [OPTION]... [TARGET]...

- GNU utilities to maintain groups of program
- Automatically determine which part of large program needs to be recompiled
  - Make update a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist
- Efficient compilation
- Take a Makefile to specify all target and prerequisite

Makefile Example

```
# Makefile - A Basic Example
all : shop  #usually first
shop : item.o shoppingList.o shop.o
        g++ -g -Wall -o shop item.o shoppingList.o shop.o
item.o : item.cpp item.h
        g++ -g -Wall -c item.cpp
shoppingList.o : shoppingList.cpp shoppingList.h
        g++ -g -Wall -c shoppingList.cpp
shop.o : shop.cpp item.h shoppingList.h
        g++ -g -Wall -c shop.cpp
clean :
        rm -f item.o shoppingList.o shop.o shop
```

Rule

Comments

Targets

Prerequisites

Dependency Line

Commands

Introduction to Python

- High-level

  (high readability, not efficient as C)
- General-purpose
- Interpreted
- Dynamic (dynamic type system)
- Automatic memory management
- Also support Object-oriented programming
  - Support class
  - Support member function

Python List

- Common data structure in Python
- A python list is like a C array but much more
  - **Dynamic**: expands as new items are added
  - **Heterogeneous:** can hold objects of different types
- Access elements: List_name[index]
- Example

```
>>> t = [123, 3.0, 'hello!']
>>> print t[0]          –123
>>> print t[1]          – 3.0
 >>> print t[2]          hello!
```

List Operations

- >>> list1 = [1, 2, 3, 4]
- >>> list2 = [5, 6, 7, 8]
- Adding an item to a list
  - list1.append(5)
  - Output: **[1, 2, 3, 4, 5]**
- Merging lists
  - >>> merged_list = list1 + list2
  - >>> print merged_list
  - Output: **[1, 2, 3, 4, 5, 5, 6, 7, 8]**

Python Dictionary

- Essentially a hash table
  - Provides key-value (pair) storage capability
- Instantiation:
  - dict = {}
  - This creates an EMPTY dictionary
- Keys are unique, values are not!
  - Keys must be immutable (strings, numbers, tuples)

Example

- dict = {}
- dict[‘hello’] = “world”
- print dict[‘hello’]
  - world
- dict[‘power’] = 9001
- if (dict[‘power’] > 9000):
- print “It is over “, dict[‘power’]
  - It is over 9001
- del dict[‘hello’]
- del dict

for loops

```
list = [‘Mary’, ‘had’, ‘a’, ‘little’, ‘lamb’]
```

```
for item in list:
    print item
```

Result:

Mary
had
a
little
lamb

```
for i in range(len(list)):
    print i
```

Result:

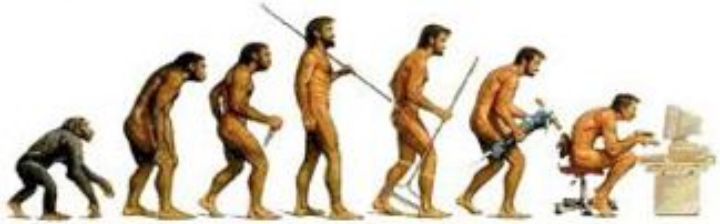
0
1
2
3
4

Indentation

- Python has no braces or keywords for code blocks
  - C delimiter: {}
  - bash delimiter:
    - then…else…fi (if statements)
    - do…done (while, for loops)
- Indentation makes all the difference
  - Tabs change code's meaning!!

A more powerful Environment

- Higher mammals use advanced tools !



- Anaconda
  - An data science platform powered by python
  - Support Different OS(Windows, Mac OS, Linux)
  - Easy to use
  - Powerful Tools (Jupyter notebook)

  https://www.continuum.io/downloads

Running Python scripts

- Use the example in randlin.py
- Make sure it has executable permission:

  chmod +x randline.py

- Run it

  ./randline.py –n 2 filename

  n: is an option indicating the number of lines to write

  2: is an argument to n (you can use any number)

  Filename: is a program argument

```
#!/usr/bin/python
```

Tells the shell which interpreter to use

```
import random, sys
from optparse import OptionParser
```

Import statements, similar to include statements
Import OptionParser class from optparse module

```
class randline:
  def __init__(self, filename):
  f = open (filename, 'r')
  self.lines = f.readlines()
  f.close ()

  def chooseline(self):
  return random.choice(self.lines)
```

The beginning of the class statement: randline
The constructor
Creates a file handle
Reads the file into a list of strings called lines
Close the file

The beginning of a function belonging to randline
Randomly select a number between 0 and the size of lines and returns the line corresponding to the randomly selected number

```
def main():
    version_msg = "%prog 2.0"
    usage_msg = """%prog [OPTION]...
FILE Output randomly selected lines from
FILE."""
```

The beginning of main function

```
parser = OptionParser(version=version_msg,
usage=usage_msg)
parser.add_option("-n", "--numlines",
action="store", dest="numlines",
default=1, help="output NUMLINES lines
(default 1)")
options, args =
parser.parse_args(sys.argv[1:])
try:
    numlines = int(options.numlines)
except:
    parser.error("invalid NUMLINES: {0}".
format(options.numlines))
if numlines < 0:
    parser.error("negative count: {0}".
format(numlines))
if len(args) != 1:
    parser.error("wrong number of operands")
input_file = args[0]
try:
    generator = randline(input_file)
    for index in range(numlines):
        sys.stdout.write(generator.chooselin
e())
except IOError as (errno, strerror):
    parser.error("I/O error({0}): {1}".
format(errno, strerror))
if __name__ == "__main__":
    main()
```

Creates OptionParser instance

Start defining options, action "store" tells optparse to take next argument and store to the right destination which is "numlines". Set the default value of "numlines" to 1 and help message.

options: an object containing all option args

args: list of positional args leftover after parsing options

Try block

get numline from options and convert to integer

Exception handling

error message if numlines is not integer type, replace {0} w/ input

If numlines is negative

error message

If length of args is not 1 (no file name or more than one file name)

error message

Assign the first and only argument to variable input_file

Try block

instantiate randline object with parameter input_file

for loop, iterate from 0 to numlines – 1

print the randomly chosen line

Exception handling

error message in the format of "I/O error (errno):strerror

In order to make the Python file a standalone program

Homework 3

comm.py

Homework 3

- comm.py – this should end up working almost exactly like the utility 'comm'
  - Check $ man comm for extensive documentation
- Extra option –u
  - Means input files are not required to be pre-sorted
  - Could sort them, but then have to maintain original ordering
    - Other ways to accomplish this?

Comm.py

- Use randline.py as a start point
- Support all options for comm

–-1, -2, -3 and combinations

–Extra option –u for comparing unsorted files

- Support all type of arguments

–File names and – for stdin

- If you are unsure of how something should be output, run a test using existing comm utility!

–Create your own test inputs

Reference

- Optparse tutorial

https://docs.python.org/2/library/optparse.html

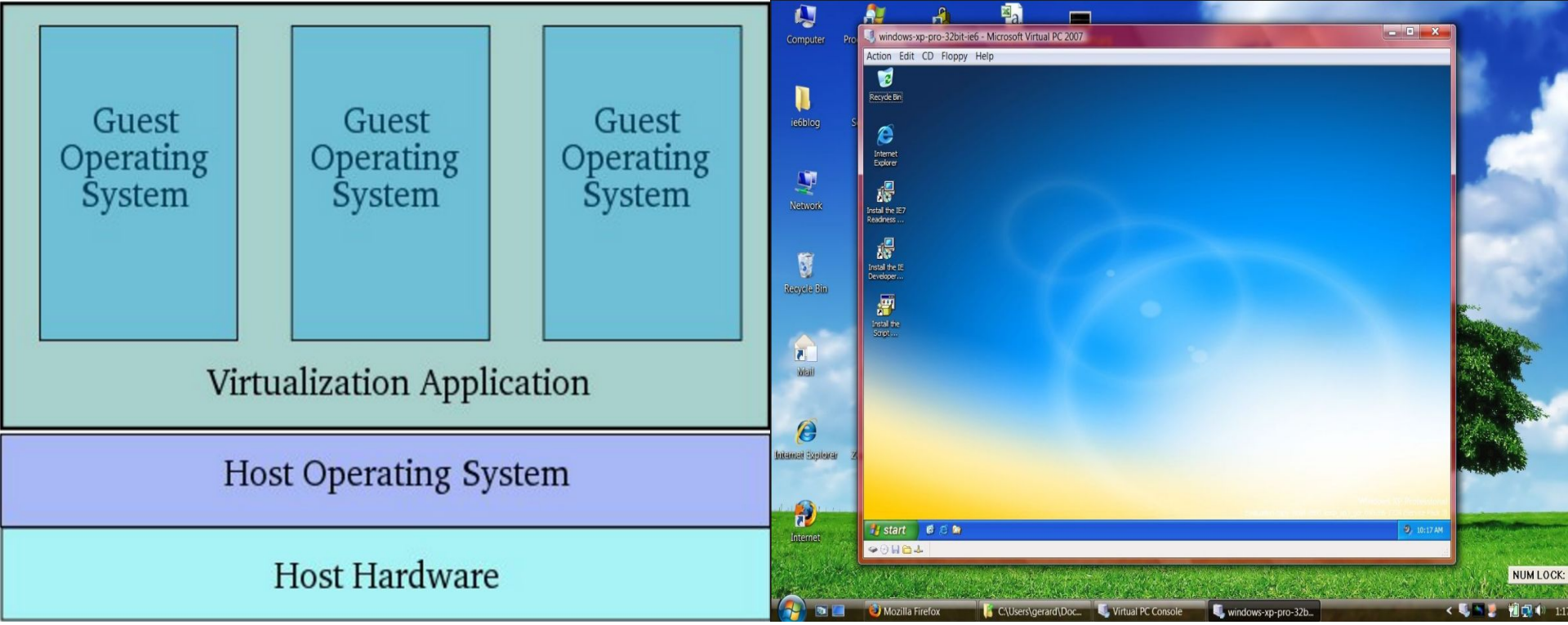
- Python tutorial

https://docs.python.org/3/tutorial/

Virtual Machine

- A program that acts like a virtual computer
- Runs on host OS and provides virtual hardware to guest OS
- Guest OS runs in a window on host OS, just like any other program
- From user's perspective, guest OS seems to be running on a physical machine
- Some popular virtual machine software:

  VirtualBox, VMvare



Some applications

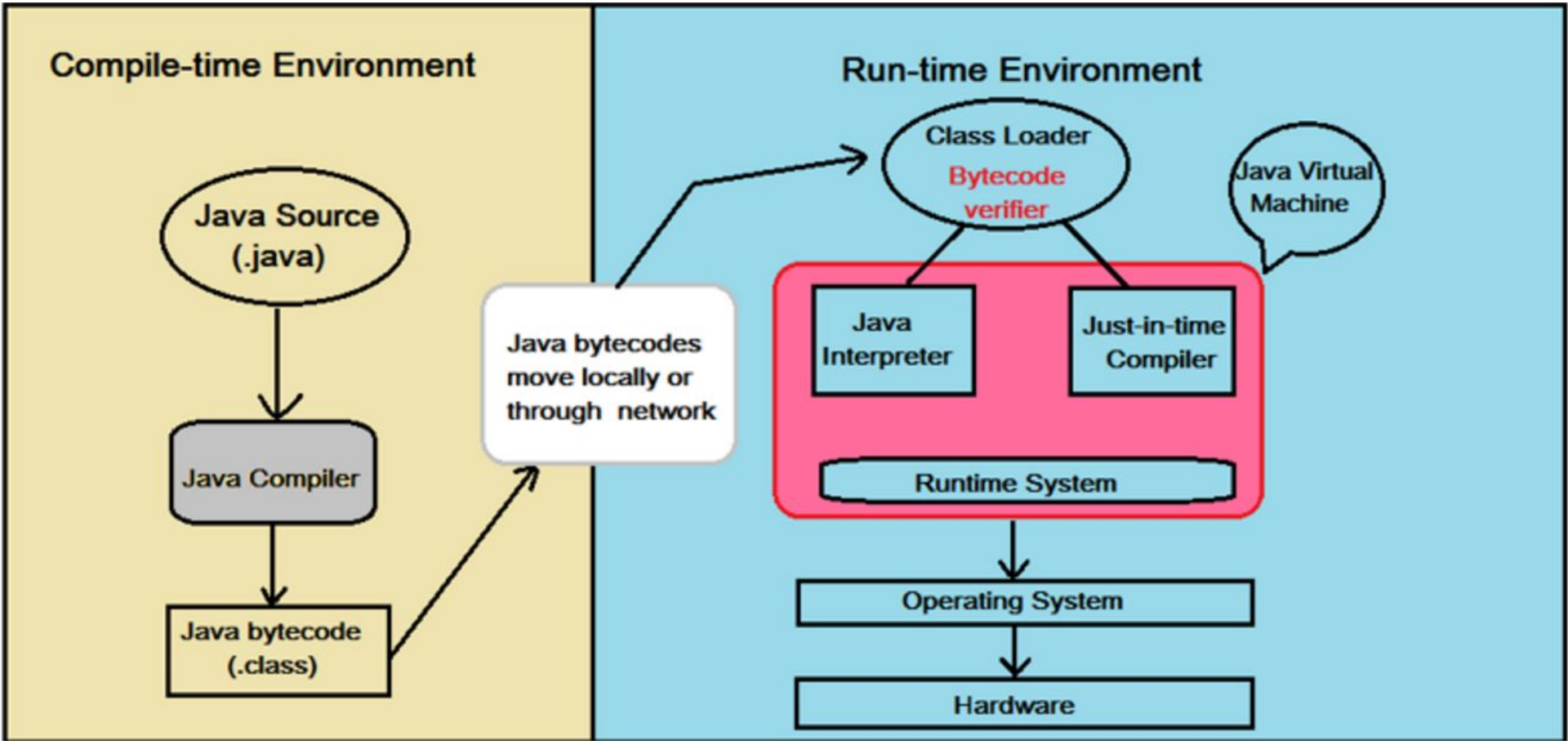
- Experiment with other OS
- Test software on multiple platforms
- Consolidate servers

Java virtual machine (JVM)

- An abstract computing machine that enables a computer to run a java program
- JVM is OS specific
- JVM interprets .class (bytecode) file and converts it to machine specific instruction set
- JVM provides a platform independent way of executing code, which makes java very portable

Two-step compilation process

- First stage: java code is compiled down to bytecode by java compiler (javac)
- Second stage: bytecode is then interpreted or compiled to run by JVM depending on the implementation of JVM


Compile-time Environment
Java Source
(.java)
Java Compiler
Java bytecode
(.class)
Java bytecodes
move locally or
through network
Run-time Environment
Class Loader
Bytecode
verifier
Java Virtual
Machine
Java
Interpreter
Just-in-time
Compiler
Runtime System
Operating System
Hardware

Java basics

- Java as a compromise between interpreted and compiled language
- JRE (Java runtime environment)

  JVM + Java class libraries
- JDK (Java Development Kit)

  JVM + Java class libraries + java compiler

CS 35L Software Construction Lab
Week 4 – C Programming

Basic Data Types

- **int**
  - Holds integer numbers
  - Usually 4 bytes
- **float**
  - Holds floating point numbers
  - Usually 4 bytes
- **double**
  - Holds higher-precision floating point numbers
  - Usually 8 bytes (double the size of a float)
- **char**
  - Holds a byte of data, characters
- **void**

Pointers

- Variables that store memory addresses

Declaration

- <variable_type> *<name>;
  - int *ptr; //declare ptr as a pointer to int
  - int var = 77; // define an int variable
  - ptr = &var; // let ptr point to the variable var

Dereferencing Pointers

- Accessing the value that the pointer points to

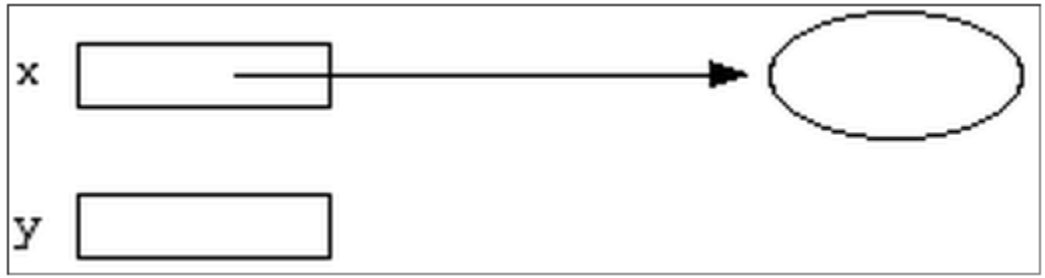
- Example:
  - double x, *ptr;
  - ptr = **&**x; // let ptr point to x
  - *****ptr = 7.8; // assign the value 7.8 to x

Pointer Example

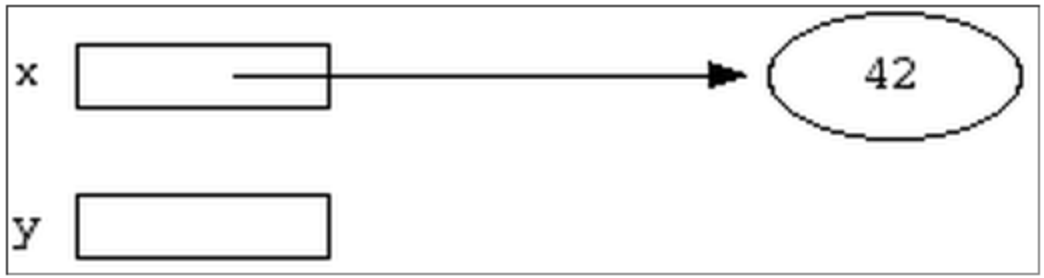
int *x;

int *y;

int var; x = &var;

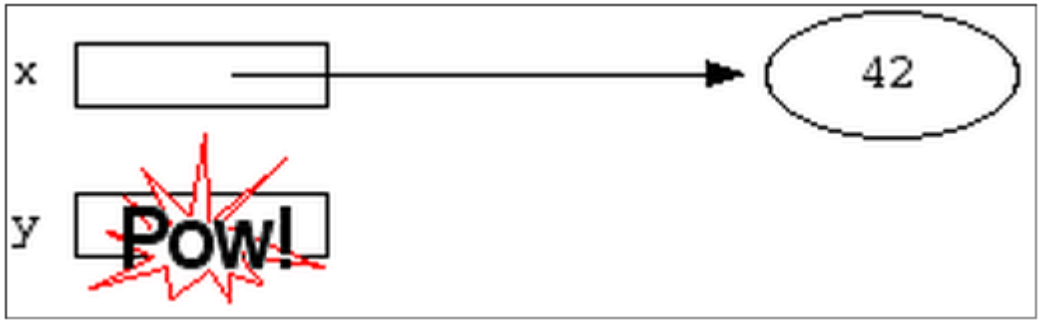


*x = 42;

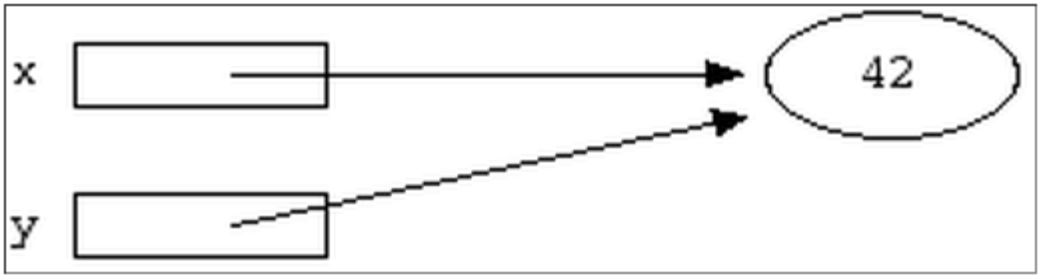


Pointer Example

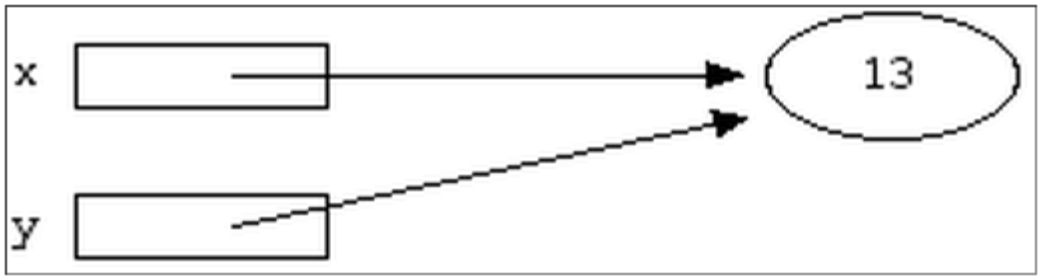
*y = 13;



y = x;

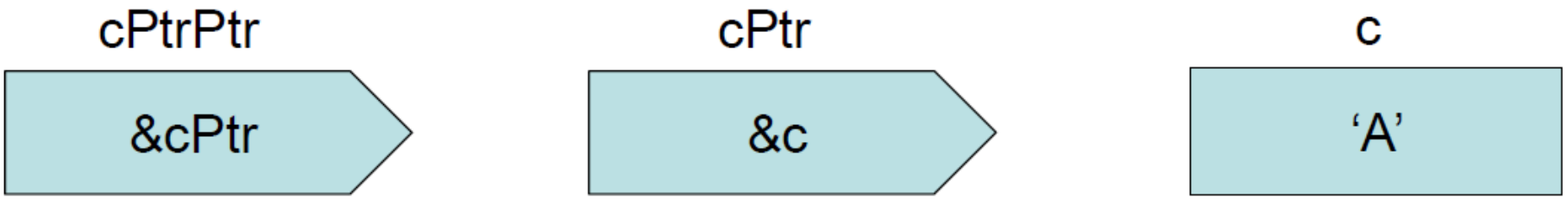


*x = 13; or
*y = 13;



Pointers to Pointers

char c = ‘A’ char *cPtr = &c char **cPtrPtr = &cPtr



Pointers to Functions

- Also known as: **function pointers** or **functors**
- Goal: write a sorting function
  - Has to work for ascending and descending sorting order + other
- How?
  - Write multiple functions
  - Provide a flag as an argument to the function
  - Use function pointers!!

Pointers to Functions

- User can pass in a function to the sort function
- Declaration
  - double (*func_ptr) (double, double);
  - func_ptr = &pow; // func_ptr points to pow()
  - func_ptr = pow; // an alternative way
- Usage
  - // Call the function referenced by func_ptr

    double result = (*func_ptr)(1.5, 2.0);
  - // an alternative way

    double result = func_ptr(1.5, 2.0);

qsort Example

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

int compare (const void * a, const void * b)
 {
        return ( *(int*)a - *(int*)b );
}

int main ()
{
        int values[] = { 40, 10, 100, 90, 20, 25 };
        qsort (values, 6, sizeof(int), compare);
        int n;
        for (n = 0; n < 6; n++)
                printf ("%d ",values[n]);
        return 0;
}
```

Structs

- No classes in C
- Used to package related data (variables of different types) together
- Single name is convenient

```
struct Student {
        char name[64];
        char UID[10];
        int age;
        int year;
};
struct Student s;
```

```
typedef struct {
        char name[64];
        char UID[10];
        int age;
        int year;
    } Student;
    Student s;
```

C structs vs. C++ classes

- C structs cannot have member functions
- There's no such thing as access specifiers in C
- C structs don't have constructors defined for them

- C++ classes/structs can have member functions
- C++ class members have access specifiers and are **private** by default
- C++ classes must have at least a default constructor

Dynamic Memory

- Memory that is allocated at runtime
- Allocated on the heap

void *malloc (size_t size);

  - Allocates *size* bytes and returns a pointer to the allocated memory
  - Allocates space in the heap during the execution of the program.
  - does not initialize the allocated memory . It carries garbage value.
  - returns null pointer if it cannot allocate requested amount of memory.

void* calloc (size_t num, size_t size);

- Allocates a block of memory for an array of num elements, each of them size bytes long
- initializes all its bits to zero.

void *realloc (void *ptr, size_t size);

- Changes the size of the memory block pointed to by *ptr* to *size* bytes
- expanding or contracting the existing area pointed to by ptr, if possible.
- OR allocating a new memory block of size new_size bytes, copying memory area with size equal the lesser of the new and the old sizes, and freeing the old block.

void free (void *ptr);

- Frees the block of memory pointed to by *ptr*
- If a null pointer is passed as argument, no action occurs.

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

int main(void)
{
    int *pa = malloc(10 * sizeof *pa); // allocate an array of 10 int
    if(pa) {
        printf("%zu bytes allocated. Storing ints: ", 10*sizeof(int));
        for(int n = 0; n < 10; ++n)
            printf("%d ", pa[n] = n);
    }

    int *pb = realloc(pa, 1000000 * sizeof *pb); // reallocate array to a larger size
    if(pb) {
        printf("\n%zu bytes allocated, first 10 ints are: ", 1000000*sizeof(int));
        for(int n = 0; n < 10; ++n)
            printf("%d ", pb[n]); // show the array
        free(pb);
    } else { // if realloc failed, the original pointer needs to be freed
        free(pa);
    }
}
```

Reading/Writing Characters

- int getchar();
  - Returns the next character from stdin
- int putchar(int character);
  - Writes a character to the current position in stdout

Formatted I/O

- int fprintf(FILE * fp, const char * format, …);
- int fscanf(FILE * fp, const char * format, …);
  - FILE *fp can be either:
    - A file pointer
    - stdin, stdout, or stderr
  - The format string
    - int score = 120; char player[] = “Mary”;
    - fp = fopen(“file.txt”, “w+”)
    - fprintf(fp, **“%s has %d points.\n”, player, score**);

Homework 5

- Write a C program called *sfrob*
  - Reads stdin byte-by-byte **(getchar)**
    - Consists of records that are newline-delimited
  - Each byte is frobnicated (XOR with dec 42)
    - Sort records without decoding (**qsort, frobcmp**)
    - Output result in frobnicated encoding to stdout **(putchar)**
  - Dynamic memory allocation (**malloc, realloc, free**)

Example

- Input: printf 'sybjre obl’
  - $ printf 'sybjre obl**\n**' | ./sfrob
- Read the records: sybjre, obl
- Compare records using *frobcmp* function
- Use *frobcmp* as compare function in *qsort*
- Output: obl

  sybjre

Homework Hints

- Array of pointers to char arrays to store strings (char ** arr)
- Use the right cast while passing frobcmp to qsort
  - cast from void ** to char ** and then dereference because frobcmp takes a char *
- Use realloc to reallocate memory for every string and the array of strings itself, dynamically
- Use *exit*, not *return* when exiting with error

CS 35L Software Construction Lab
Week 4 – Debugging

Debugging Process

- Reproduce the bug
- Simplify program input
- Use a debugger to track down the origin of the problem
- Fix the problem

Debugger

- A program that is used to run and debug other (target) programs
- Advantages:

  Programmer can:

  – step through source code line by line
    - each line is executed on demand
  – interact with and inspect program at run-time
  – If program crashes, the debugger outputs where and why it crashed

GDB – GNU Debugger

- Debugger for several languages
  - C, C++, Java, Objective-C… more
- Allows you to inspect what the program is doing at a certain point during execution
- Logical errors and segmentation faults are easier to find with the help of gdb

Using GDB

1. **Compile Program**
   - Normally: `$ gcc [flags] <source files> -o <output file>`
   - Debugging: `$ gcc [other flags] -g <source files> -o <output file>`
     - enables built-in debugging support

2. **Specify Program to Debug**
   - `$ gdb <executable>`

     or
   - `$ gdb`
   - `(gdb) file <executable>`

Run-Time Errors

- Segmentation fault
  - Program received signal SIGSEGV, Segmentation fault. 0x0000000000400524 in *function* (arr=0x7fffc902a270, r1=2, c1=5, r2=4, c2=6) at *file.c*:12
    - Line number where it crashed and parameters to the function that caused the error
- Logic Error
  - Program will run and exit successfully
- How do we find bugs?

Using GDB

3. **Run Program**
   - `(gdb) run` or
   - `(gdb) run [arguments]`
4. **In GDB Interactive Shell**
   - Tab to Autocomplete, up-down arrows to recall history
   - `help [command]` to get more info about a command
5. **Exit the gdb Debugger**
   - `(gdb) quit`

Setting Breakpoints

- Breakpoints
  - used to stop the running program at a specific point
  - If the program reaches that location when running, it will pause and prompt you for another command
- Example:
  - `(gdb) break file1.c:6`
    - Program will pause when it reaches line 6 of file1.c
  - `(gdb) break my_function`
    - Program will pause at the first line of `my_function` every time it is called
  - `(gdb) break [position] if expression`
    - Program will pause at specified position only when the expression evaluates to true

Breakpoints

- Setting a breakpoint and running the program will stop program where you tell it to

- You can set as many breakpoints as you want
  - `(gdb) info breakpoints|break|br|b` shows a list of all breakpoints

Basic commands

- `(gdb)step` - Step to next line of code. Will step into a function.
- `(gdb)print <var>` - Print value stored in variable.
- `(gdb)next` - Execute next line of code. Will not enter functions.
- `(gdb)continue` - Continue execution to next break point.
- `(gdb)set var <name>=<value>` - Executes rest of program with new value of variable.

Deleting, Disabling and Ignoring BPs

- `(gdb) delete [bp_number | range]`
  - Deletes the specified breakpoint or range of breakpoints
- `(gdb) disable [` *`bp_number | range`*`]`
  - Temporarily deactivates a breakpoint or a range of breakpoints
- `(gdb) enable [` *`bp_number | range`*`]`
  - Restores disabled breakpoints
- If no arguments are provided to the above commands, all breakpoints are affected!!
- `(gdb) ignore` *`bp_number iterations`*
  - Instructs GDB to pass over a breakpoint without stopping a certain number of times.
    - bp_number: the number of a breakpoint
    - Iterations: the number of times you want it to be passed over

Displaying Data

- Why would we want to interrupt execution?
  - to see data of interest at run-time:
  - `(gdb) print [/format]` *`expression`*
    - Prints the value of the specified expression in the specified format
  - Formats:
    - d: Decimal notation (default format for integers)
    - x: Hexadecimal notation
    - o: Octal notation
    - t: Binary notation

Resuming Execution After a Break

- When a program stops at a breakpoint
  - 4 possible kinds of gdb operations:
    - **c or continue**: debugger will continue executing until next breakpoint
    - **s or step**: debugger will continue to next source line
    - **n or next**: debugger will continue to next source line in the current (innermost) stack frame
    - **f or finish**: debugger will resume execution until the current function returns. Execution stops immediately after the program flow returns to the function's caller
      - the function's return value and the line containing the next statement are displayed

Watchpoints

- Watch/observe changes to variables
  - `(gdb) watch my_var`
    - sets a watchpoint on my_var
    - the debugger will stop the program when the value of *my_var* changes
    - old and new values will be printed
  - `(gdb) rwatch` *`expression`*
    - The debugger stops the program whenever the program reads the value of any object involved in the evaluation of *expression*

Process Memory Layout

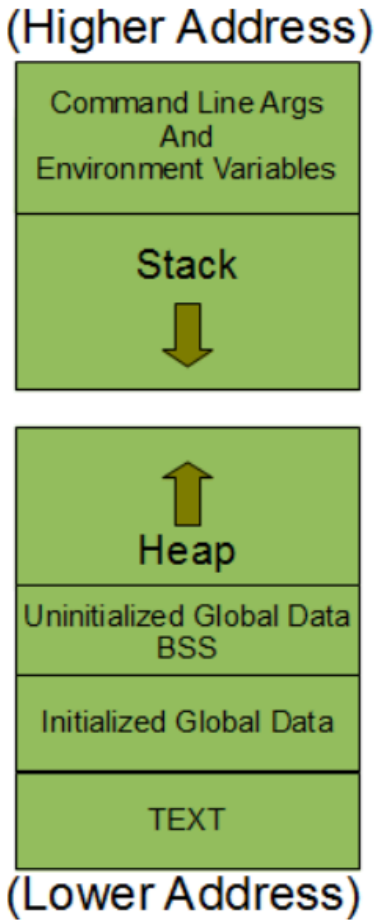


Image source : thegeekstuff.com

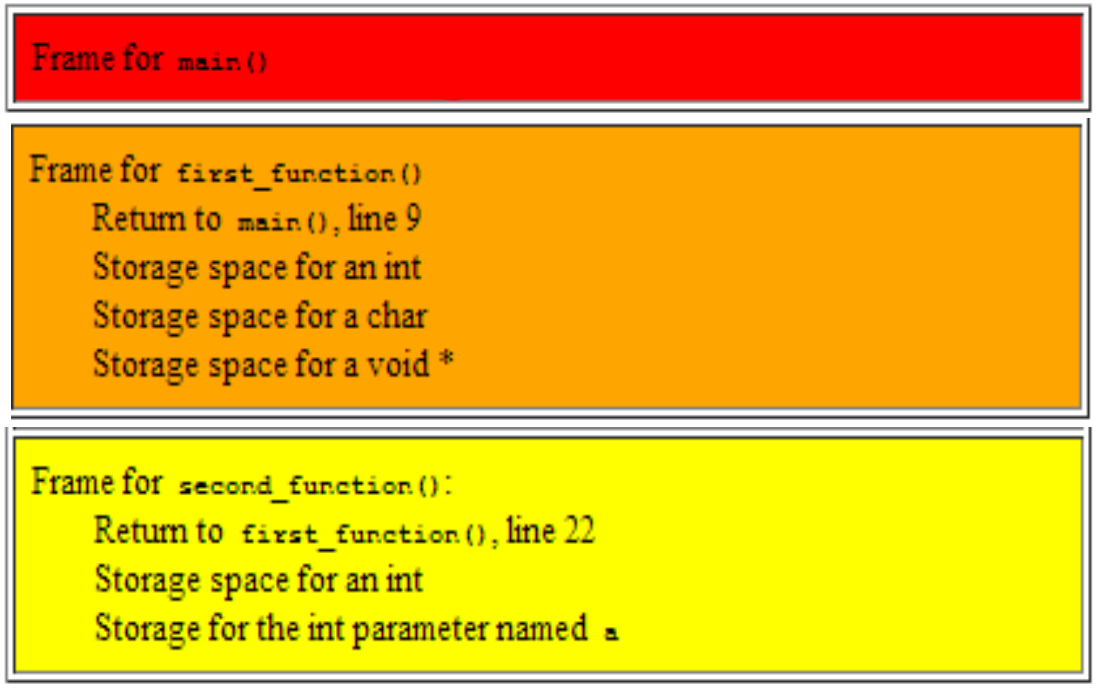
- TEXT segment
  - Contains machine instructions to be executed
- Global Variables
  - Initialized
  - Uninitialized
- Heap segment
  - Dynamic memory allocation
  - malloc, free
- Stack segment
  - Push frame: Function invoked
  - Pop frame: Function returned
  - Stores
    - Local variables
    - Return address, registers, etc
- Command Line arguments and Environment Variables

Stack Info

- A program is made up of one or more functions which interact by calling each other
- Every time a function is called, an area of memory is set aside for it. This area of memory is called a **stack frame** and holds the following crucial info:
  - storage space for all the local variables
  - the memory address to return to when the called function returns
  - the arguments, or parameters, of the called function
- Each function call gets its own stack frame. Collectively, all the stack frames make up the **call stack**

Stack Frames and the Stack

```
1   #include <stdio.h>
2   void first_function(void);
3   void second_function(int);
4
5   int main(void)
6   {
7      printf("hello world\n");
8      first_function();
9      printf("goodbye goodbye\n");
10
11     return 0;
12  }
13
14
15  void first_function(void)
16  {
17     int imidate = 3;
18     char broiled = 'c';
19     void *where_prohibited = NULL;
20
21     second_function(imidate);
22     imidate = 10;
23  }
24
25
26  void second_function(int a)
27  {
28     int b = a;
29  }
```



Analyzing the Stack in GDB

- `(gdb) backtrace|bt`
  - Shows the call trace (the call stack)
  - Without function calls:
    - #0 main () at program.c:10
    - one frame on the stack, numbered 0, and it belongs to main()
  - After call to function display()
    - #0 display (z=5, zptr=0xbffffb34) at program.c:15

      #1 0x08048455 in main () at program.c:10
    - Two stack frames: frame 1 belonging to main() and frame 0 belonging to display().
    - Each frame listing gives
      - the arguments to that function
      - the line number that's currently being executed within that frame

Analyzing the Stack

- (gdb) info frame
  - Displays information about the current stack frame, including its return address and saved register values
- (gdb) info locals
  - Lists the local variables of the function corresponding to the stack frame, with their current values
- (gdb) info args
  - List the argument values of the corresponding function call

Other Useful Commands

- (gdb) info functions
  - Lists all functions in the program
- (gdb) list
  - Lists source code lines around the current line

Lab 4

- Download old version of coreutils with buggy ls program
  - Untar, configure, make
- Bug: ls -t mishandles files whose time stamps are very far in the past. It seems to act as if they are in the future

```
$ touch -d '1918-11-11 11:00 GMT' wwi-armistice
$ touch now
$ sleep 1
$ touch now1
$ ls -lt wwi-armistice now now1
```

Output:

-rw-r--r-- 1 eggert eggert 0 Nov 11 1918 wwi-armistice

-rw-r--r-- 1 eggert eggert 0 Feb 5 15:57 now1

-rw-r--r-- 1 eggert eggert 0 Feb 5 15:57 now

Goal: Fix the Bug

- **Reproduce the Bug**
  - Follow steps on lab web page
- **Simplify input**
  - Run ls with –l and –t options only
- **Debug**
  - Use gdb to figure out what's wrong
  - $ gdb ./ls
  - (gdb) run –lt wwi-armistice now now1

  (run from the directory where the compiled ls lives)
- **Patch**
  - Construct a patch "lab5.diff" containing your fix
  - It should contain a ChangeLog entry followed by the output of diff -u

Lab Hints

- Don’t forget to answer all questions! (lab4.txt)
- Make sure not to submit a reverse patch! (lab4.diff)
- “Try to reproduce the problem in your home directory, instead of the $tmp directory. How well does SEASnet do?”
  - Timestamps represented as seconds since Unix Epoch 1970.1
  - SEASnet NFS filesystem has unsigned 32-bit time stamps
  - Local File System on Linux server has signed 32-bit time stamps
  - If you touch the files on the NFS filesystem it will return timestamp around 2054
  - => files have to be touched on local filesystem (df –l)
- Use “info functions” to look for relevant starting point
- Compiler optimizations: -O2 -> -O0
  - ./configure CFLAGS="…-O0"