CS197U: A Hands on Introduction to Unix

Lecture 7: Version Control and Advanced Topics

Tian Guo
University of Massachusetts Amherst – CICS

Reminders

- Questions ?
  - tian@cs.umass.edu
- Assignment 4 is due on Oct 15.
  - Let me know if you’re skipping this.
- Have your own virtual machine configured !
  - Will need to be a root in the following lectures
- No Class on Oct 13 (Monday Schedule)
- A Guest Lecture on Oct 15
- No office hours on Oct 15

Last time

- File Processing Utilities
  - Awk: Quick reviews and examples
  - Sed
- Shell Scripting
  - automating command line tasks

Today

- **Programming in Linux**
  - compilers and accessories
- Version Control
  - helps synchronize files shared with multiple users
- Advanced Vim
- Other Useful Commands

Programming in Linux

- Compiled languages - larger programs, optimized performance
  - C, C++, C#
  - Java

- Scripting languages - ease of development, cross platform
  - Shell scripting
  - Python
  - Perl
  - Ruby

C / C++ programming

- `gcc` and `g++` are the main C / C++ compilers
- Usage: `gcc file.c -o execname`
  - If you don't use -o flag, default name is "a.out"
- For projects with multiple files:

```
g++ -c file1.cpp      # produces file1.o
g++ -c file2.cpp      # produces file2.o

# link the .o files into an executable
g++ file1.o file2.o -o execname
```

```
g++ file1.cpp file2.cpp -o execname
```

Building big programs

- If you are installing a program from source code or writing big apps...
- Don’t want to manually compile and link each file and library
- **Makefiles** simplify the build process
  - Contains a list of “build targets”
  - Run `make <target>` to automate compilations steps
- Common procedure when installing from source:
  - `./configure` - checks that you have libraries, prepares configuration files
  - `make` - compiles the main application code
  - `make install` - copies the new executables to their final paths
    - (may require sudo!)

Simple Makefile example

```
# A simple Makefile

myprogram: main.o library.o
	gcc -o myprogram main.o library.o

library.o: library.c library.h header.h
	gcc -c library.c

main.o: main.c header.h
	gcc -c main.c

clean:
	@rm -f myprogram main.o library.o
```

make myprogram
will automatically build the other “targets”

Make tutorial: http://mrbook.org/blog/tutorials/make/

Other languages

- Java
  - Compile to create .class files:
    - javac file.java
    - javac file1.java file2.java file2.java
  - Run: java file (where file.java contains a main method)
- Python
  - type ‘python’ at command prompt to use an interactive interpreter
  - Run script by: python filename.py
- Perl/Ruby
  - Run script by: perl filename.pl or ruby filename.rb

Today

- **Programming in Linux**
  - compilers and accessories
- **Version Control**
  - helps synchronize files shared with multiple users
- Advanced Vim
- Other Useful Commands

Version control systems

- Used to coordinate files being edited by multiple people
- Or just use it for your own projects to track history
- Helps resolve conflicts from simultaneous edits
- Stores history of files so you can easily revert / branch projects

- Many different version control systems:
  - CVS
  - Subversion
  - Git
  - Mercurial
  - etc, etc

Version control concepts

- **Repository** - The set of files or directories that are under version control
  - Stores the tracking history
  - You never edit these files directly
- **Local copy** - “check out” files from repository to edit them
  - Modify the files as you like
- **Update** - updates your local copies
  - Detects if another user modified the same file
  - Attempts to merge changes (only works for text files)
- **Commit** - sends your versions of the files to the repository
  - Will only update repository if there are no conflicts
  - A collection of actions that are performed inside the repository

Subversion workflow

- “Checkout” a working copy of the repository
  - `svn checkout svn+ssh://<USERNAME>@elnux7.cs.umass.edu/<PATH>`
  - Creates a local copy of all files
- Work on the files in the working copy.
  - Everyone works on their own local copies
- Add new files to be tracked
  - `svn add <file or directory>`

Subversion workflow

- Commit your changes back to the repository
  - `svn commit`
  - Double checks that there are no new conflicts
  - Uploads your changes
- Update the files to see other people's changes
  - `svn update`
  - Applies their changes to your local files
  - Will warn you about conflicts
    - Must manually resolve files with edits to same lines
- Now your partner will have to get your changes
  - `svn update`

Subversion Commands

```
> svn checkout svn+ssh://<USERNAME>@elnux7.cs.umass.edu/<PATH>
## makes you a local copy of the files

> svn commit -m “A message describing what you’ve done”
## sends your files to the repository

> svn add <filename or directory>
## Adds a file or directory to local copy and will send to repository
on next commit

> svn status
## shows status of files (A – added, M – modified, ? – not tracked, D-
deleted etc.)

> svn list svn+ssh://<USERNAME>@elnux7.cs.umass.edu/<PATH>
## prints out the files contained in repository

> svn update
## updates your files
```

SVN on Edlab machines

- Each student has their own **empty** SVN repository in their course home directory
  - `/courses/cs100/cs197u/<username>/svn`

- You should not interact directly with this directory
  - `ls svn`
    - `conf, db, format, hooks, locks, README.txt`
  - **Do not add/remove files from here**

SVN on Edlab machines

- Instead, use svn checkout command to create a local copy in another directory
  - When you check out a copy, it will also be called ‘svn’ on your machine
    - Usually the repository is stored somewhere else, but on the Edlab machines this setup works best because of permissions
  - You can interact with these files, make new files (on your machine)
    - They are only modified on your machine
- Use svn commands to add/remove files to/from the repository
  - Then **`svn commit`** them, they’ll be updated on the repository
  - You will get a revision number for this round

Why use version control?

- Track and back up versions of your code
  - Revert to an older version quickly and easily
  - **`svn revert`** file_name
    - Revert to a previous version if it is not yet committed
- Share files with a group
  - Make individual changes and distribute code to everyone

Outline

- Programming in Linux
  - compilers and accessories
- Version Control
  - helps synchronize files shared with multiple users
- **Advanced Vim**
- Other Useful Commands

Important Vim commands

- There are three commonly used modes in Vim.
  - Normal: return to this mode with ESC
  - Insert: Start inserting new text; from normal mode, press “i” to enter
  - Visual: Text selections; from normal mode, press “v” to enter
- Save without closing file: ESC + :w
- Save and quit: ESC + :wq
- Vim cheat sheet: http://vim.rtorr.com/
- Advanced Vim Tips: http://vim.wikia.com/wiki/Vim_Tips_Wiki

vi or **vim** - advanced text editor #2

- Mode based - probably opposite of what you'd expect
  - Normal Mode - keys invoke different commands
  - Insert Mode - keys type characters to the screen

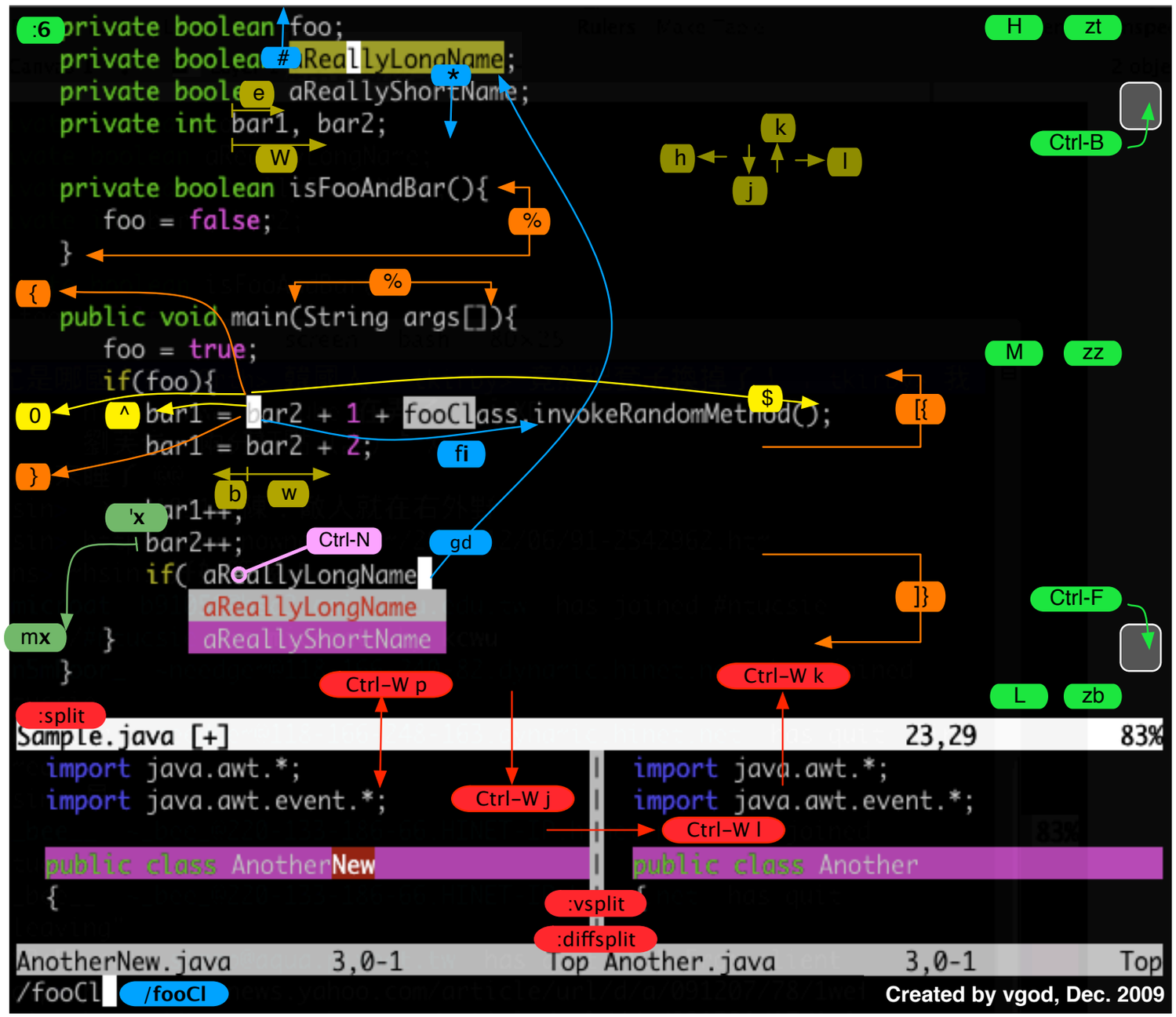
```
Press i to enter INSERT mode
   - must be in insert mode to type new text
   - move around with arrow keys (generally)

Press Esc to return to NORMAL mode
   - type to run commands
      :w     = save file (write to disk)
      :q     = quit
      :wq    = save and quit
      :q!    = quit without saving
~
~
~
-- INSERT --                                  11,1          All
```

dozens of other commands. Some start with ":" some don't

vim filename

vim Cheat Sheet

Vim Visual Cheat Sheet



Movement/Range

Character

h j k l

word, WORD(all non-blank ch)

w b	next/prev word
W B	next/prev WORD
e E	end of word/WORD

Line

0 $	begin/end of line
^	begin (non-blank) of line

Paragraph, Block

{ }	prev/ next paragraph
[{]}	begin/end of block
%	matching parenthesis

Window, File

H	top of win
zt	scroll to top
M	mid of win
zz	scroll to middle
L	btm of win
zb	scroll to bottom
C-B C-F	prev/next page
gg G	begin/end of file
mx 'x	mark/jump to x

Search

* #	find current word backward/forward
fx	to character x to right
gd	to definition of current word
/xxx	search xxx
n N	next/prev search result

Mode Commands

ESC C-[	enter normal mode
v	enter visual mode
V	enter visual line mode
C-v	enter visual block mode
i	enter insert mode
R	enter replace mode
a	append
A	append at end of line

General Commands

y	yank/copy (range)
d	delete/cut (range)
c	modify (range)
x	delete/cut (character)
D	delete to end of line
C	modify to end of line
p	paste after cursor
J	join lines
r	replace (character)
>	indent
<	indent leftward
.	redo
u	undo

EX Commands

:w	save(:wq save and quit)
:q	quit(:q! quit anyway)
:e x	edit file x
:n	new window
:h	vim help
:xx	jump to line #xx

Auto-completion [insert mode]

C-N C-P	auto-complete next/prev keyword
C-X C-F	auto-complete file name

Split window

:vsp :sp	vertically/horizontally split
:diffs	split and diff
C-W p	to last accessed window
C-W w	to next window

Configuring vim using vimrc

- syntax on/off: toggle color syntax
- set number/nonumber: toggle on/off line number
- noh: remove the highlight by searching

- Use pathogen.vim to install and manage vim plugins
  - https://github.com/tpope/vim-pathogen

Outline

- Programming in Linux
  - compilers and accessories
- Version Control
  - helps synchronize files shared with multiple users
- Advanced Vim
- **Other Useful Commands**

Suspend, background, foreground

- To suspend a job (pause)
  - Type **ctrl-z**
  - suspends the current job, returns to prompt
- To check status of jobs
  - Type **jobs**

```
[1]+  Stopped                 bc
```

- To run a job in the background/foreground
  - Type **bg/fg %<job #>: fg %1**
  - No arguments runs the most recently suspended job
  - **<command> &** runs a job in the background directly

screen

- Program that allows you to detach/attach to terminal sessions
  - Useful for running long processes
    - Start process, detach from terminal session, log out of machine
    - log in to machine, attach to same terminal session, view results

`screen -S <name>`	Starts a new screen session
`screen -r <name>`	Attaches to an existing
`ctrl-a d`	Detach from session
`ctrl-d`	Terminate session
`screen -ls`	Lists current screen sessions
`exit (in screen session)`	Closes screen session

`nohup` – detaching processes

- Usually, you run a command and wait for its output
  - Can background a job and do other stuff
  - all your running jobs get killed when logging out/disconnected

- `nohup <command-with-flags> &`
  - standard output goes to `nohup.out` unless other file is given
  - Will run the command in the background, detached from terminal
  - close and reopen the terminal, will be still running (if not finished yet)!

`cron` – scheduling re-occurring jobs

- Run a command regularly
  - e.g. every day, every hour, on weekdays
  - `crontab -e`
    - Edit cron file
    - `<min> <hr> <day_of_month> <month> <day_of_week> <command>`
    - One `job' per line (make sure there is a new empty line for new job)
- `crontab -l`
  - Display listing of cron jobs
- Examples
  - `00 18 * * 1-5 mail -s "work on CS197u assignment" me@me.com`
  - `00 00 1 1 * echo "HAPPY NEW YEAR!"`
  - `00 7 * * * backup.sh`

Scheduling jobs after booting up

- Run a command after booting up
- There are several ways to do this
  - Edit `/etc/rc.local`
    - Make sure it is executable
    - This is run as root!
  - Add a line to your cron file
    - `@reboot /path/to/shell.script`
    - This is run without root access
- More crontab examples
  - http://www.thegeekstuff.com/2009/06/15-practical-crontab-examples/

Display Forwarding

- Use `ssh -X` flag to forward the graphics from programs to your local machine
  - `ssh -X` username@elnux.cs.umass.edu

- Example
  - When you open Matlab, the GUI version displays
  - Try typing **gedit**, a text editor window will pop up

`rsync`

- Synchronizes files and directories
  - locally or across a network
  - great for backing up files
- `rsync -avz  <source> <destination>`
  - `-a` archive
  - `-v` verbose
  - `-z` compress
  - `-P` show progress and keep partially transferred files
    - Resume from partial transfer possible
- --delete to reflect deletions in source in the destination directory
- Only copies files that change after first backup, much faster

Looking for things

- `find -` prints the path to a file (also remember `locate`)
  - **`find <dir> -name 'assignment_*'`**

- `whereis -` locate a binary file
  - `> whereis perl`
    - `/usr/bin/perl`

- `which`
  - `> which ls`
    - `/bin/ls`

`diff, cd`

- `diff` - compares two files
  - `diff <file1> <file2>`
  - Output shows lines that appear in one file and lines numbers in both files
- `‘cd -’` - moves to the last directory you were in

Next week

- We’ll focus on network system
- Background knowledge of UNIX networking
- Basic skills to control/monitor network traffic of your system