CS250: DISCUSSION #2

Brian Zimmer

9/8/2011

OVERVIEW

- Logistics
- Unix
- Makefiles
- Git
- Python

LOGISTICS

- Lab #1 is due on Monday at 1pm
- Make sure your work is both committed AND pushed to github
- You will have two weeks for Lab #2

EDITORS

- I highly recommend using Vim
- Emacs also ok
- Please try one of these, it will help in the long run

UNIX OVERVIEW

- The tools we use consume and generate text as their principle inputs and outputs
  - Some binary formats exist, but since many people use tools from different companies, it becomes important to have open formats
- Our class will use bash as the shell of choice, many use C-Shell (csh, tcsh)
- Tools are setup using environment variables...need to understand how these work

ENVIRONMENT VARIABLES AND SETTINGS

- Path: Where to look for binaries
  - See path: `echo $PATH` (PATH is an environment variable)
  - Add to path: `export PATH=/some/path:$PATH`
  - `source ~cs250/tools/cs250.bashrc`
- Alias: Shortcuts
  - `alias ..="cd ../"`
- Functions
  - `function cd { builtin cd "$@" && ls -F }`
- .bashrc vs .bash_profile
  - `.bash_profile` should contain paths and will include `.bashrc`
  - `.bashrc` should contain alias's, functions

BASH

- Shortcuts
  - Ctrl-a: move to start of line
  - Ctrl-e: move to end of line
  - Ctrl-u: Delete from cursor to beginning of line
  - Ctrk-k: Delete from cursor to end of line
  - Ctrl-p: Repeat last command
  - Alt-r: Undoes changes to line
- Command line
  - `!$` gets replaced with last argument of last command
    - eg. `cp local.txt /some/other/long/dir/copy.txt; vim !$`
  - `^search^replace` will replace typo in last command
    - eg. `vim /some/path/wrongfile; ^wrongfile^rightfile`

UNIX

- Symbolic links: `ln -s targetfile linkname`
- Compare two files: `diff fileA fileB` (or `vimdiff fileA fileB`)
- Compare two directories: `diff -rq dirA dirB`
- Search for text in directory: `grep "searchstring" *`
- Search for text recursively: `find . | xargs grep searchstring` OR if ack is installed, `ack searchstring`
- Search for a filename: `find . -name "*.pdf"`
- Kill process: `ps aux | grep processname; kill processid` OR `top`; "`u`" and enter username, then "`k`" and enter process id
- Quick search and replace in a file: `sed -i 's:searchstring:replacestring:g' filename` (drop the -i to preview the result)
- Search and replace recursively in a directory: `find * | xargs perl -pi -e 's/find/replace/g'`
  - "`find *`" can also be "`ack -l`" (after doing an "ack" to make sure you are only changing what you want)
  - "`perl -pi -e`" can also be "`sed -i`"
- Temporarily change directories: `pushd /path/to/another/dir`, then to return to original directory, "`popd`"

TRICK: FORWARD TO NX

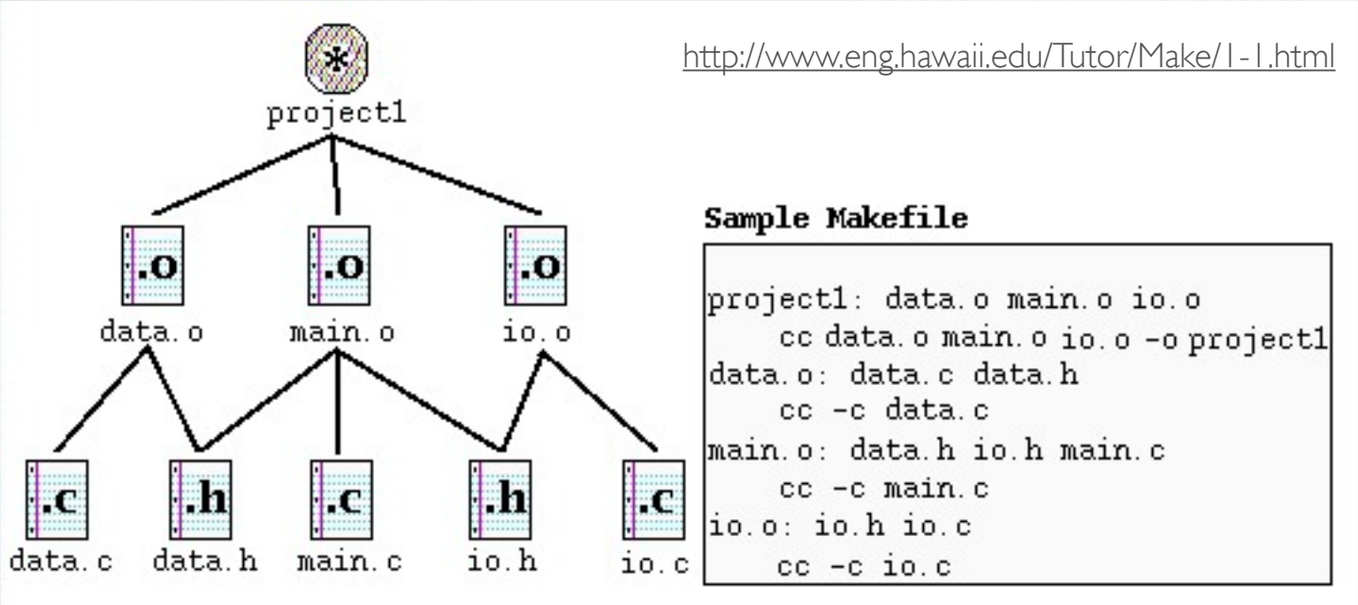
- In `.bashrc`:
  - `alias getnx="echo \"export DISPLAY=$DISPLAY\" > ~/.nxdisplay"`
    `alias setnx="source ~/.nxdisplay"`
- Then in terminal inside NX: `getnx`
- Then in terminal without X: `setnx`

TRICK: STORE COMMAND LINE ARGUMENTS

- Slow:
  - `somecommand -a blah -b blahblah -d / somepath/ -xsf ...`
- Fast:
  - `vim args` type them in here
  - `somecommand `cat args``

MAKEFILES

- Each line is a command, rule, or variable
  - Rule: `target: dependency` (commands belonging to this rule must be indented with a tab)
  - Variable: `varname = varvalue` (= is recursive, := is expanded right away so can reference itself or something that changes)
- Reference the value of a variable with `$(variablename)`
- Use `\` to continue a long line
- # for comments
- Need to use actual tabs, not spaces
- eg. `build/dc-syn/Makefile`
- Will only compile a given rule if its dependency source files are newer



REGULAR EXPRESSIONS

- From campus internet connection (or VPN), safaribooksonline.com: Mastering Regular Expressions
- Useful in scripts and editors
- Syntax is different for every program (eg. Perl vs. Python vs. egrep)
  - Some major differences (eg. looking across multiple lines)
- Learn as needed (Google helps), keep track of snippets used

SOME REGULAR EXPRESSION BASICS

- **^** matches start of line
- **$** matches end of line
- **.** matches any one character
- **[ ]** matches one of any of the characters inside (eg. [abc])
- **( )** matches string of inside (eg. (cat|dog))
- Quantifiers: eg.
  - **?** means character ahead is optional
  - ***** (0 or more) of proceeding character
  - **+** (1 or more) of proceeding character
  - **{n}** matches n
- **^** negates (eg. [^u] is any character except u)
- **** turns metacharacter into literal (escapes it)

EXAMPLES

- Match a string within double quotes: “`[^"]*`”
- Find whitespace at the end of a line (use \s to match a space): `\s$`

REGULAR EXPRESSIONS REPLACE

- `()blah()` remembered with `\1` and `\2`
- Different platforms
  - Vim: Where I recommend doing it first...you can see the matches, undo
    - `:%s/search/replace/g` in all lines (:s/ means current line, or do visual selection then press :s/)
    - /gc instead of /g will allow you to confirm each
    - "very magic"...way less escaping. Manually "/\v" instead of "/" or ":%s/\v" instead of ":%s/"
  - Python
    - ```
      import re
      outvar = re.sub("search","replace",invar)
      ```
  - Sed
    - `sed -i "s:search:replace:g"` (/ works as well) (in OS X, `sed -i "" -e "s:search:replace:g"` instead)
- Examples
  - Wrap every word in quotes:
    - Vim: `:%s/\([^ ]\+\)/"\1"/g` or in very magic mode: `:%s/\v([^ ]+)/\1/g`
      Python: `outvar = re.sub("([^ ]+)","\"\\1\"",invar)`
      ```

GIT

- Read the tutorial and associated references
- Try using the github.com interface
- Anyone have any problems?

PYTHON

- Sometimes we need something more complicated than bash scripts and makefiles to automate things
- Perl also ok
- Save a history of code snippets to use in the future
- Basics: Indentation instead of matching brackets

PYTHON LIST

```
#! /usr/bin/python

alist = []
alist.append(1)
alist.append(10)
alist.append(3)

print alist

alist[1] = 9

for aelement in alist:
  print aelement

def mul2(x):
  return 2*x

alist = map(mul2, alist)

for aelement in alist[:2]:
  print aelement
```

PYTHON READ FILE

```
#! /usr/bin/python

import sys
import string

if len(sys.argv) < 2:
  print "usage: %s <file>" % sys.argv[0]
  sys.exit(1)

f = open(file)
lines = map(string.strip, f.readlines())
f.close()

for line in lines:
  print line
```

PYTHON LINE SPLIT

```
#! /usr/bin/python

import sys
import string

if len(sys.argv) < 2:
  print "usage: %s <file>" % sys.argv[0]
  sys.exit(1)

f = open(file)
lines = map(string.strip, f.readlines())
f.close()

for line in lines:
  strs = line.split()
  for str in strs:
    print str

for i,line in enumerate(lines):
   strs = lines[i+2].split()
```

PYTHON FIND STRING

```
#! /usr/bin/python

import sys
import string

if len(sys.argv) < 2:
  print "usage: %s <file>" % sys.argv[0]
  sys.exit(1)

f = open(file)
lines = map(string.strip, f.readlines())
f.close()

for line in lines:
  if line.find('search string') != -1:
     print line
```

IF TIME...

- Circuit review
- Work in groups on writing the python needed in Lab 1