COMP2004 Programming Practice 2002 Summer School

Kevin Pulo
School of Information Technologies
University of Sydney

More Unix Tools

- Keeping track of code versions
- More Unix shell scripting
  - Including common Unix utilities

Revision Control

- Keeping track of changes to code
- Not using one leads to
  - Accidently deleting all your code
  - Breaking your working code
- Often also supports group work
  - Merging changes
  - Access control

Simple revisioning

- cp -r src src.bak1
- cp -r src src.bak2
- ...
- Extremely simple
  - Possibly adequate for small code
- Wastes space - copies all src / obj files
  - Better to just save differences
- Easy to accidentally delete a backup
- Doesn't support groupwork

PRCS

- Easy to setup and use revision control
- Using it will make coding easier
- There's information on it in the tute
- ~sholden/pub/prcs/bin/prcs
- CVS is a more complicated/powerful system

The shell

- The shell is actually a programming language
- It has if, for, while, etc statements
- It has variables and operators
- Can run other programs and manipulate their input/output
- Many common tasks often done by other Unix utilities

Simple commands I

- cd - Change directory
- cp - Copies files/dirs
- mv - Moves (renames) files/dirs
- rm - Removes (deletes) files/dirs

- Common parameters:
  - -f = force
  - -r = recursive
  - -i = interactive (ask before overwriting)

Simple commands: echo

- echo - Prints parameters to stdout

```
bash$ echo "Hello World"
Hello World
bash$
```

- Options (for bash's echo):
  - -n - No trailing newline
  - -e - Allow \n, \t, etc

Simple commands: cat

- cat - Prints files (or stdin) to stdout

```
bash$ cat hello.txt
Hello World
bash$
```

Redirection

- Output:
  - echo "Hello World" > hello.txt
- Appending:
  - echo -ne "Hello\t" > hello.txt
  - echo "World" >> hello.txt
- Input:
  - cat < hello.txt

Redirection caveat

- Eg: Suppose a program called prog:
  - Reads from file.txt
  - Does some processing
  - Outputs to stdout
- prog file.txt > file.txt
- Contents of file.txt are deleted by the shell before prog is run!
- Thus prog reads an empty file
- And then outputs nothing
- So file.txt stays empty

Redirection caveat

- Solution: use a temporary file instead

```
prog file.txt > file.tmp
mv -f file.tmp file.txt
```

Exit Status

- Returned by a program when it exits
- That's why main() returns an int
- 0 is value for success (use EXIT_SUCCESS in C++)
- Everything else is failure
- cmp exits with 0 for identical files
- We can use this to help out testing

If in shell

- If tests the exit value of a command
- 0 is true, everything else is false
- Recall that
  - Our testing output is in outfile
  - Correct output is in test.out

```
if cmp test.out outfile; then
        echo "test passed"
else
        echo "test failed"
fi
```

elif

```
if cmp test.out outfile; then
        echo "test passed"
elif diff -i test.out outfile; then
        echo "test passed (only just)"
else
        echo "test failed"
fi
```

- The -i option to diff ignores case
  - ie: a more forgiving comparison

Exit in shell

- Can control the exit status of a shell script with exit <num>
- Eg:
  - exit
  - exit 0
  - exit 1
- Useful when the command in an if statement is another shell script:

```
if another_script; then
    ...
fi
```

Testing variables

- String-based
  - if ["$somevar" = "hello"]; then
  - if ["$somevar" != "hello"]; then
- Numeric-based
  - if ["$num" -lt 42]; then
  - if ["$num" -ge 42]; then
- File-based
  - if [-r "$fname"]; then
  - if ["$fname" -nt "file.txt"]; then
- Many more, for info: man test

Putting stdout into variables

- Sometimes want to put stdout from a program into a shell variable
- Use backticks:

```
echo "f1.h f2.h f3.h" > file.lst
filelist=`cat file.lst`
cat $filelist
cat `cat file.lst`
```

- bash has alternate syntax

```
cat $(cat file.lst)
```

Shell arithmetic

- Use a command called expr

```
bash$ expr 3 + 8 \* 2
19
bash$
```

- bash has alternate syntax, instead of
  - prog `expr 3 + 8 \* 2`
- can use
  - prog $((3 + 8 * 2))
- which is equivalent to
  - prog 19

while

- Standard counting loop:

```
count=1
while [ $count -le 10 ]; do
        echo "$count"
        count=$(($count + 1))
done
```

while

- Infinite loop:

```
while :; do
        if [ -e "file.txt" ]; then
                break
        fi
        sleep 1
done
```

Command line parameters

- The command line parameters are $1, $2, $3, ..., $9
- bash also supports ${10}, ${11}, ...
- Name of the shell script is $0
- Number of parameters is $# (no $0)
- shift - deletes $1, makes $2 be $1, $3 be $2, etc
- shift n - shifts n times

Example parameters

- The shell script sample:

```
#!/gnu/usr/bin/bash
while [ $# -ge 1 ]; do
        echo "$1"
        shift
done
```

```
bash$ sample a bc defgh
a
bc
defgh
```

case

```
case "$1" in
        -d*)
                cmd="diff"
                ;;
        -c*)
                cmd="cmp"
                ;;
        *)
                exit 1
                ;;
esac
$cmd file1.txt file2.txt
```

case

- Using diff:
  - compare -d
  - compare -diff
  - compare -djofiwni
- Using cmp:
  - compare -c
  - compare -cmp
  - compare -compare
- Cause fail exit:
  - compare -e
  - compare 42
  - compare

Testing script

- Testing script so far:

```
#!/gnu/usr/bin/bash
prog < test.in > outfile
if cmp test.out outfile; then
        echo "test passed"
else
        echo "test failed"
fi
```

Pipes

- You can redirect one program's output to another's input
- This removes the need for outfile

```
if prog < test.in | cmp - test.out; then
        echo "test passed"
else
        echo "test failed"
fi
```

Multiple Tests

- Multiple tests are always needed
- We can automate them as well
- Put all our tests in a directory (tests)
- Have the input files named xyz.in
- Have the output files name xyz.out
- Where xyz is the test name
- We can use the shell to run them all

Running Multiple Tests

```
#!/gnu/usr/bin/bash
for infile in tests/*.in; do
        name=`basename $infile .in`
        if prog < $infile | cmp - \
                        tests/$name.out; then
                echo "$name : passed"
        else
                echo "$name : failed"
        fi
done
```

sed

- The Stream EDitor
- sed file.txt
  - Same as cat file.txt
- sed -e '10q' file.txt
  - Output first 10 lines
- sed -n -e '10,20p' file.txt
  - Output lines 10 to 20
- sed -e 's/foo/bar/g' file.txt
  - Replace all 'foo' with 'bar'
- More info: man sed

awk

- Extremely useful utility
- Yet another programming language
- Still good for simple tasks
- Excellent at extracting columns
- GNU gawk has many good extensions
- More info: man awk or man gawk
- Eg: output logins for summer students:

```
awk -F: '/summer semester/ {print $1}' \
    /etc/passwd
```

Other useful utilities

- sort - sorts lines in a file
- uniq - removes duplicate lines
- head - output first n lines
- tail - output last n lines
- cut - extracts columns of characters
- join, paste - merges files by columns

- For all, get more info with
  - man command