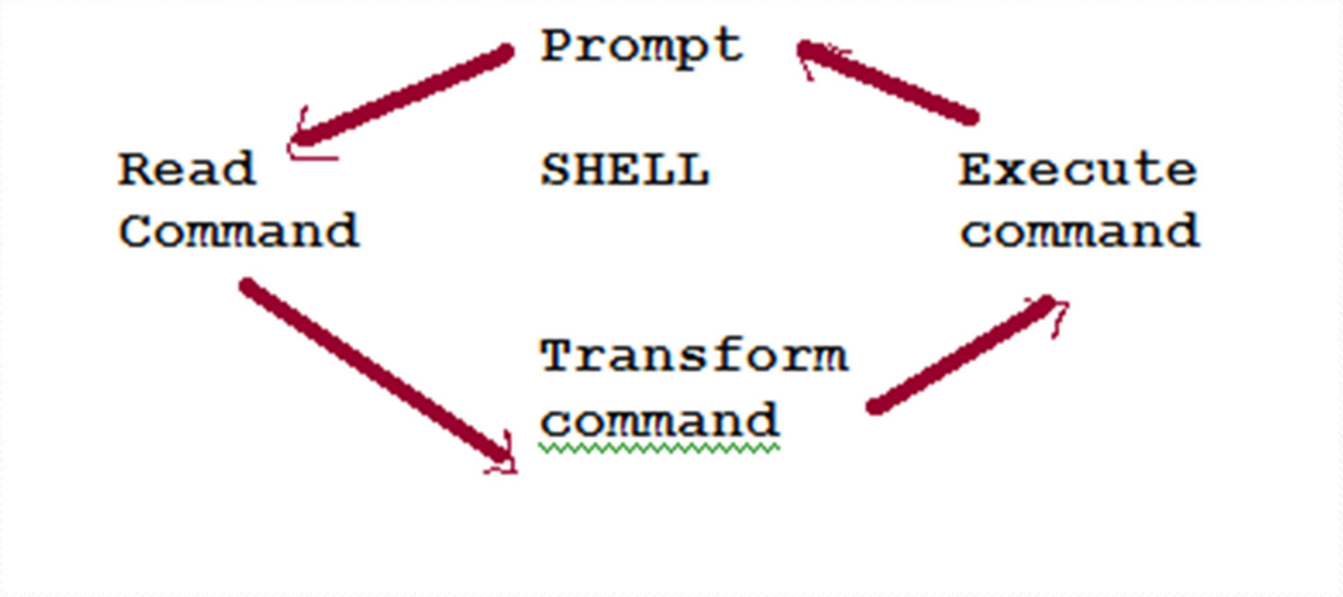
Shell Programming

15-123

Systems Skills in C and Unix

The Shell

- A command line interpreter that provides the interface to Unix OS.



What Shell are we on?

- **echo $SHELL**
- Most unix systems have
  - Bourne shell (**sh**)
    - No command history
  - Korn shell (**ksh**)
    - Shell functions
  - C shell (**csh**)
    - History, no shell functions
- More details at unix.com

What’s Shell good for?

- Starting and stopping processes
- Controlling the terminal
- Interacting with unix system
- Solving complex problems with simple scripts
  - Life saver for system administrators
- *What is a “shell script” ?*
  - A collection of shell commands supported by control statements
  - Shell scripts are interpreted and instructions executed

Quick review of basics

A Shell Script

#!/bin/sh

-- above line should always be the first line in your script

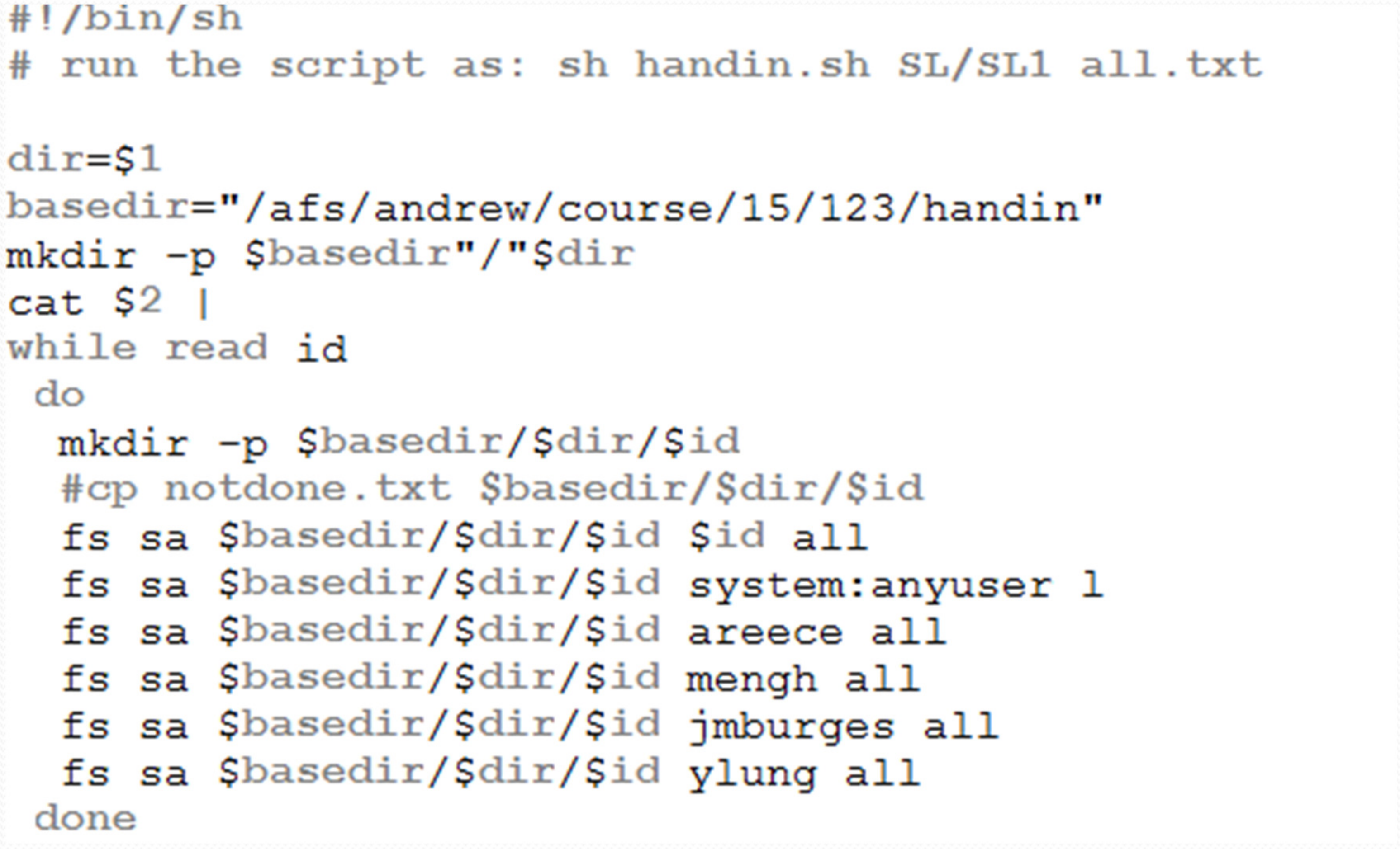
# A simple script

who am I

Date

- *Execute with: sh first.sh*

Another shell script



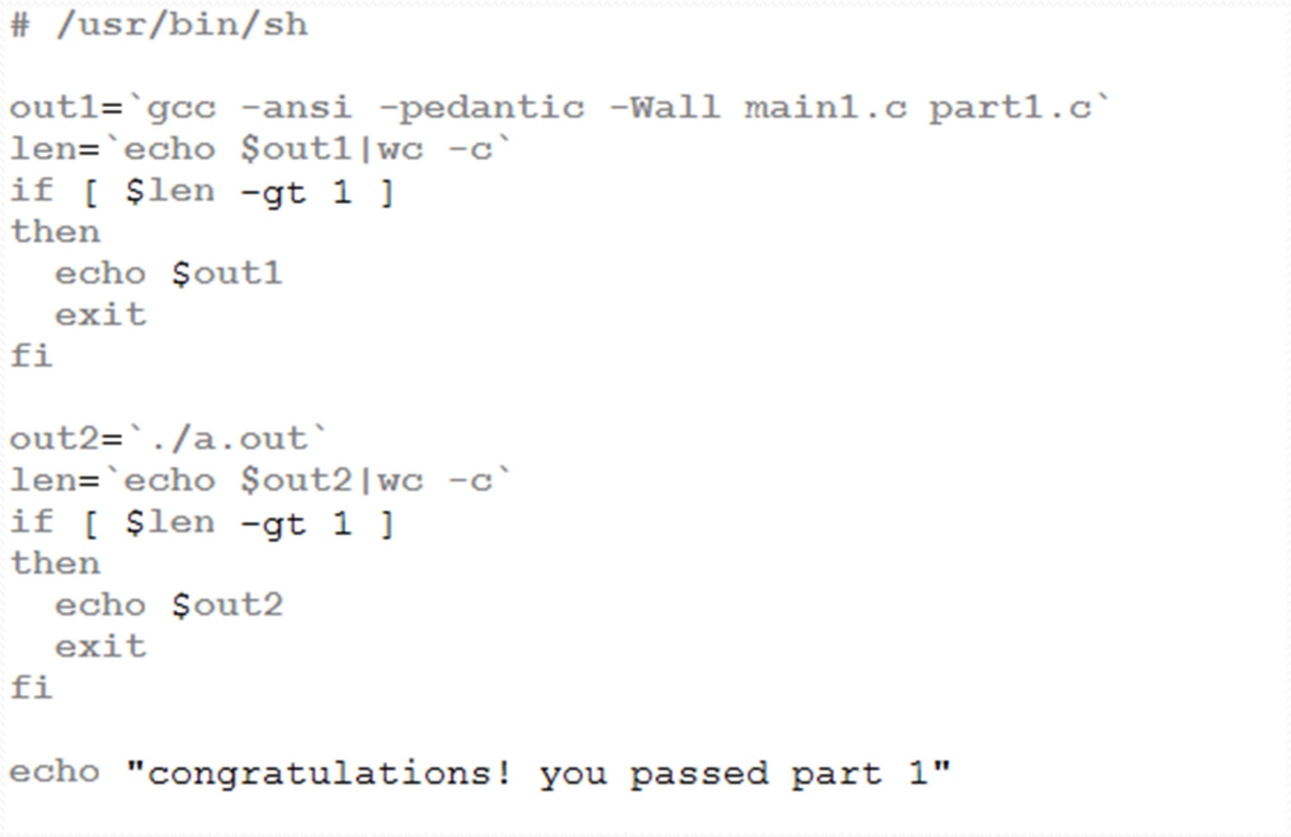
```
#!/bin/sh
# run the script as: sh handin.sh SL/SL1 all.txt

dir=$1
basedir="/afs/andrew/course/15/123/handin"
mkdir -p $basedir"/"$dir
cat $2 |
while read id
 do
  mkdir -p $basedir/$dir/$id
  #cp notdone.txt $basedir/$dir/$id
  fs sa $basedir/$dir/$id $id all
  fs sa $basedir/$dir/$id system:anyuser l
  fs sa $basedir/$dir/$id areece all
  fs sa $basedir/$dir/$id mengh all
  fs sa $basedir/$dir/$id jmburges all
  fs sa $basedir/$dir/$id ylung all
 done
```

Command Line Arguments

- $# - represents the total number of arguments (much like argv) – except command
- • $0 - represents the name of the script, as invoked
- • $1, $2, $3, .., $8, $9 - The first 9 command line arguments
  - Use "shift" command to handle more than 9 args
- • $* - all command line arguments OR
- • $@ - all command line arguments

What are the three kinds of quotes in Shell expressions?

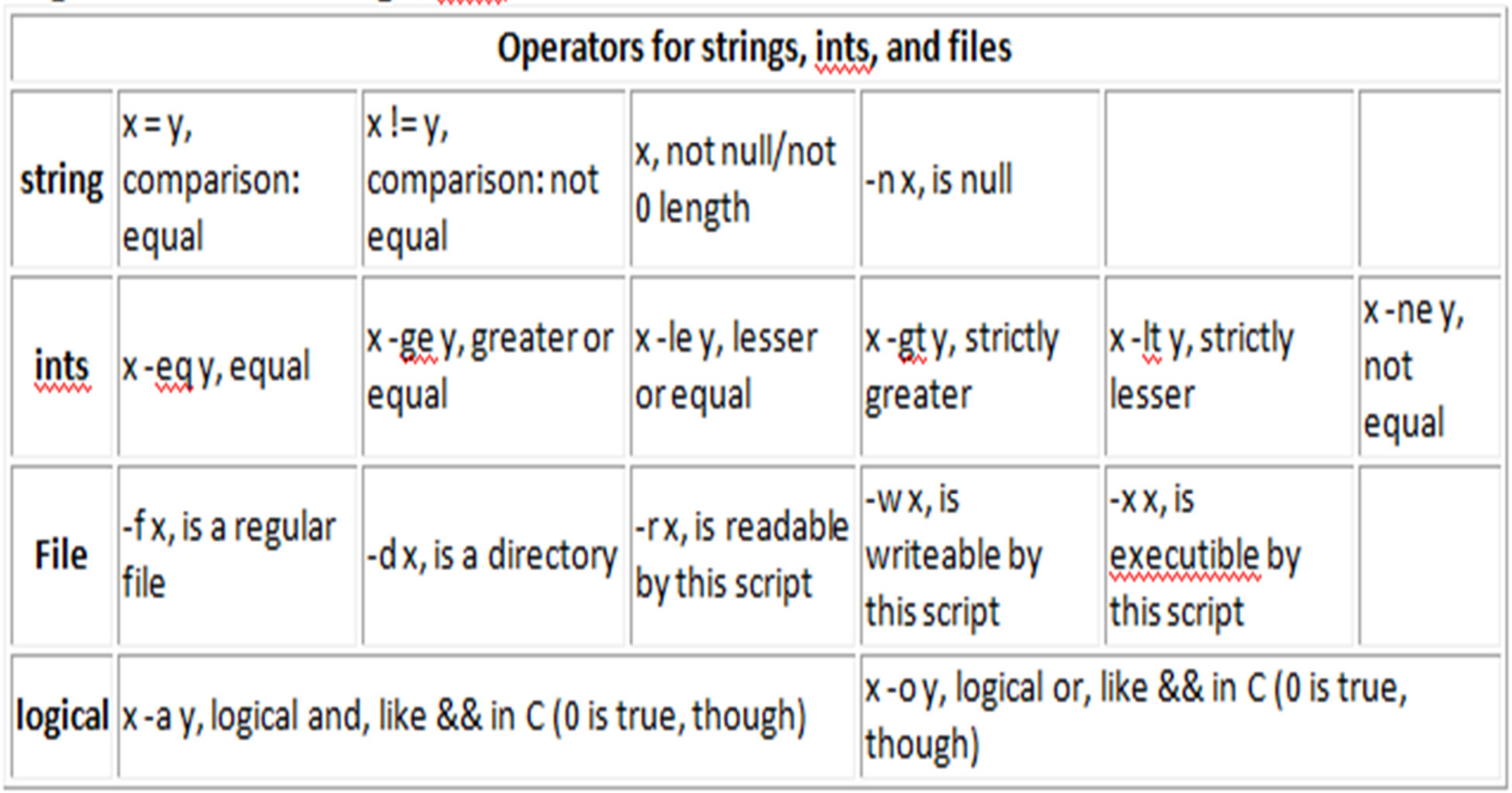
Capturing output from a shell operation



```
# /usr/bin/sh

out1=`gcc -ansi -pedantic -Wall main1.c part1.c`
len=`echo $out1|wc -c`
if [ $len -gt 1 ]
then
  echo $out1
  exit
fi

out2=`./a.out`
len=`echo $out2|wc -c`
if [ $len -gt 1 ]
then
  echo $out2
  exit
fi

echo "congratulations! you passed part 1"
```

A major bug: Did not catch if the program seg faulted

Operators for strings, ints and files



Operators for strings, ints, and files							
string	x = y, comparison: equal	x != y, comparison: not equal	x, not null/not 0 length	-n x, is null			
ints	x -eq y, equal	x -ge y, greater or equal	x -le y, lesser or equal	x -gt y, strictly greater	x -lt y, strictly lesser	x -ne y, not equal	
File	-f x, is a regular file	-d x, is a directory	-r x, is readable by this script	-w x, is writeable by this script	-x x, is executible by this script		
logical	x -a y, logical and, like && in C (0 is true, though)			x -o y, logical or, like && in C (0 is true, though)			

Control Statements – Loops and conditionals

```
for var in "$@"
   do
      printf "%s\n" $var
   done

for (( i = 1 ; i < 20 ; i++ ))
 do


done
```

```
while read file
  do
        echo $file
  done
```

```
if command
then
     command
     command
     ...
     command
else
     command
     command
     ...
     command
fi
```

```
if command
then
     command
     command
     ...
     command
fi
```

Useful shell commands

- Shell already has a collection of rich commands
- Some Useful commands
  - uptime, cut, date, cat, finger, hexdump, man, md5sum, quota,
  - mkdir, rmdir, rm, mv, du, df, find, cp, chmod, cd
  - uname, zip, unzip, gzip, tar
  - tr, sed, sort, uniq, ascii
  - Type “man command” to read about shell commands

What do these shell commands do?

- cat dups.txt | sort | uniq
- cat somefile.txt | sed 's/|/,/g' > outfile
- cat somefile.txt | sed 's#|#,#g' > outfile
- cat somefile.txt | sed '1,10 s/|/,/g' > outfile
- cat somefile.txt | sed '1,$ s/|/,/g' > outfile
- cat somefile.txt | sed '/^[0-9]+/ s/|/,/g' > outfile
- cat file | cut -d: -f3,5
- cat file.txt | tr "abcd" "ABCD" > outfile.txt

More of those

- cat file.txt | tr "a-z" "A-Z" > outfile.txt
- cat file.txt | tr -d "\015" > outfile.txt
- cat somefile.txt | tr "\015" "\012" > somefile.txt

I/O

- File descriptors
  - Stdin(0), stdout(1), stderror(2)
- Input/output from/to stdin/stdout
  - read data
  - echo $data
- redirecting
  - rm filename 1>&2

Unix tools in shell scripts

- Shell scripts can include utilities such as
  - grep
    - Pattern matching
  - sed
    - Stream editor
  - awk
    - Pattern scanning and processing
  - Read more in notes and man pages

Interprocess communication

Inter Process Communication (IPC)

- Communication between processes
- Using **Pipes**
  - Pipes is the mechanism for IPC
  - ls | sort | echo
    - 4 processes in play
- Each call spans a new process
  - Using folk
  - More later about folk

Editing in Place

- cat somefile.txt | tr -d "\015" "\012" | fold > somefile.txt
- What does it do?
- What are some of the problems?
- Problems are caused by the way pipes work

How does pipes work

- A finite buffer to allow communication between processes
  - Typically size 8K
- If input file is less than the buffer
  - We may be ok
- What if input file is more than the buffer
  - Redirecting output to the same file is a bad idea

How to deal with this?

- **Use a temp file**
  - **cat file | tr -d "\015" "\012" | fold > file.tmp**
  - **mv file.tmp file**

- **Better process**
  - cat file| tr -d "\015" "\012" | fold > "/usr/tmp/file.$$"
  - mv "/usr/tmp/file.$$" “file“
- **/usr/tmp** is cleared upon reboot

Pipes, Loops and Sub shells

```
#!/bin/sh
FILE=$1
cat $FILE  |
while read value
 do
    echo ${value}
  done
```

- while loop is executed in a sub shell

What is the problem?

```
#!/bin/sh
FILE=${1}
max=0
cat ${FILE} |
   while read value
    do
         if [ ${value} -gt ${max} ];
            then
                max=${value}
         fi
    done
echo ${max}
```

The fix

```
#!/bin/sh
FILE=${1}
max=0
values=`cat ${FILE}`
for value in ${values}
do    if [ ${value} -gt ${max} ];
   then
          max=${value}
    fi
    done
echo ${max}
```

Arrays in bash

```
array[2]=23
array[3]=45
array[1]=4
```

To dereference an array variable, we can use, for example

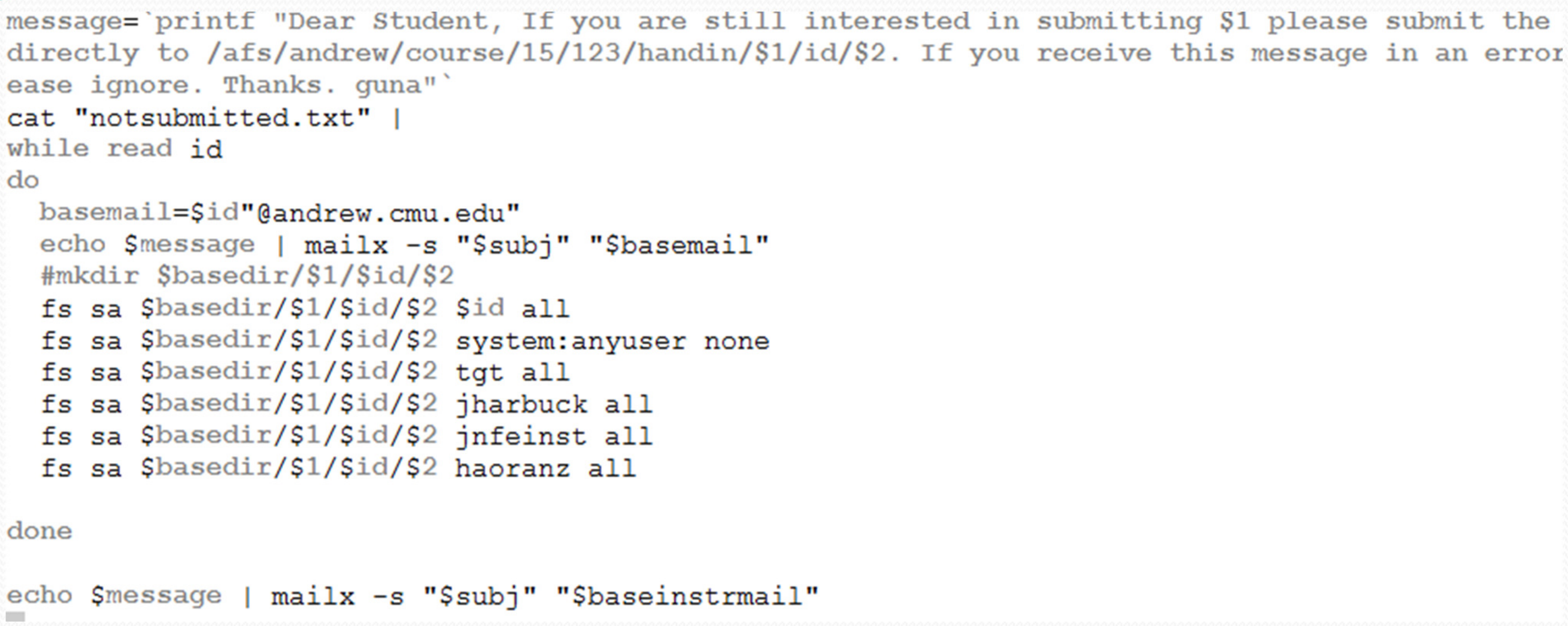
```
echo  ${array[1]}
```

Array elements need not be consecutive and some members of the array can
be left uninitialized. Here is an example of printing an array in bash. Note the
C style loop. Also note the spaces between tokens.

```
for ((  i=1  ;  i<=3  ;  i++  ))
do
  echo ${array[$i]}
done
```

Coding Examples

```
#!/bin/sh
# run the script as: sh closehandin.sh SL/SL1 all.txt

dir=$1
basedir="/afs/andrew/course/15/123/handin"

cat $2 |
while read id
 do
  fs sa $basedir/$dir/$id $id l
  fs sa $basedir/$dir/$id system:anyuser l
  fs sa $basedir/$dir/$id ylung all
  fs sa $basedir/$dir/$id areece all
  fs sa $basedir/$dir/$id jmburges all
  fs sa $basedir/$dir/$id mengh all
 done
```



```
message=`printf "Dear Student, If you are still interested in submitting $1 please submit the
directly to /afs/andrew/course/15/123/handin/$1/id/$2. If you receive this message in an error
ease ignore. Thanks. guna"`
cat "notsubmitted.txt" |
while read id
do
  basemail=$id"@andrew.cmu.edu"
  echo $message | mailx -s "$subj" "$basemail"
  #mkdir $basedir/$1/$id/$2
  fs sa $basedir/$1/$id/$2 $id all
  fs sa $basedir/$1/$id/$2 system:anyuser none
  fs sa $basedir/$1/$id/$2 tgt all
  fs sa $basedir/$1/$id/$2 jharbuck all
  fs sa $basedir/$1/$id/$2 jnfeinst all
  fs sa $basedir/$1/$id/$2 haoranz all

done

echo $message | mailx -s "$subj" "$baseinstrmail"
```