CSE 390a
Lecture 6

bash scripting continued; remote X windows; unix tidbits

slides created by Marty Stepp, modified by Josh Goodwin
http://www.cs.washington.edu/390a/

Lecture summary

- more shell scripting
  - if/else
  - while/until
  - select/case
  - advanced: arrays and functions
- Remote editing/GUI
- various new Unix/Linux commands
  - file archiving and compression
  - shell history
  - newlines in Unix vs Windows

if/else

```
if [ test ]; then                # basic if
    commands
fi

if [ test ]; then                # if / else if / else
    commands1
elif [ test ]; then
    commands2
else
    commands3
fi
```

- there ***MUST*** be a space between `if` and [and between [and ***`test`***
  - [is actually a shell command, not just a character
  - also be careful to include the semi-colon between] and `then`

Testing commands

shell command	description
=, !=, <, >	compares two string variables
-z, -n	tests whether a string is or is not empty (null)
-lt, -le, -eq, -gt, -ge, -ne	compares numbers; equivalent to Java's <, <=, ==, >, >=, !=
-e, -d	tests whether a given file or directory exists
-r, -w	tests whether a file exists and is read/writable

```
if [ $USER = "stepp" ]; then
    echo ‘Hello there, beautiful!’
fi

LOGINS=`w | wc -l`
if [ $LOGINS -gt 10 ]; then
    echo ‘attu is very busy right now!’
fi
```

More `if` testing

shell command	description
`if [ expr1 -a expr2 ]; then ...` `if [ test1 ] && [ test2 ]; then ...`	and
`if [ expr1 -o expr2 ]; then ...` `if [ test1 ] || [ test2 ]; then ...`	or
`if [ ! expr ]; then ...`	not

```
# alert user if running >= 10 processes when
# attu is busy (>= 5 users logged in)
LOGINS=`w | wc -l`
PROCESSES=`ps -u $USER | wc -l`
if [ $LOGINS -gt 5 -a $PROCESSES -gt 10 ]; then
    echo "Quit hogging the server!"
fi
```

Exercise

- Write a program that computes the user's body mass index (BMI) to the nearest integer, as well as the user's weight class:

$$BMI = \frac{weight}{height^2} \times 703$$

BMI	Weight class
≤ 18	underweight
18 - 24	normal
25 - 29	overweight
≥ 30	obese

```
$ ./bmi
Usage: ./bmi weight height

$ ./bmi 112 72
Your Body Mass Index (BMI) is 15
Here is a sandwich; please eat.

$ ./bmi 208 67
Your Body Mass Index (BMI) is 32
There is more of you to love.
```

Exercise solution

```
#!/bin/bash
# Body Mass Index (BMI) calculator
if [ $# -lt 2 ]; then
    echo "Usage: $0 weight height"
    exit 1
fi

let H2=“$2 * $2”
let BMI="703 * $1 / $H2"
echo "Your Body Mass Index (BMI) is $BMI"
if [ $BMI -le 18 ]; then
    echo "Here is a sandwich; please eat."
elif [ $BMI -le 24 ]; then
    echo "You're in normal weight range."
elif [ $BMI -le 29 ]; then
    echo "You could stand to lose a few."
else
    echo "There is more of you to love."
fi
```

Common errors

- `[: -eq: unary operator expected`
  - you used an undefined variable in an `if` test
- `[: too many arguments`
  - you tried to use a variable with a large, complex value (such as multi-line output from a program) as though it were a simple int or string
- `let: syntax error: operand expected (error token is " ")`
  - you used an undefined variable in a `let` mathematical expression

while and until loops

```
while [ test ]; do          # go while test is true
    commands
done


until [ test ]; do          # go while test is false
   commands
done

while [ "$ACTION" = "open the pod bay doors" ]; do
   echo "I'm sorry Dave, I'm afraid I can't do that."
   read -p "What would you like me to do?" ACTION
done
```

select and case

- Bash Select

```
PS3=prompt  # Special variable for the select prompt
select choice in choices; do
        commands
        # Break, otherwise endless loop
        break
done
```

- Bash Case

```
case EXPRESSION in
    CASE1) COMMAND-LIST;;
    CASE2) COMMAND-LIST;;
    ...
    CASEN) COMMAND-LIST;;
esac
```

Exercise

- Have the user select their favorite person, and output a message based on their choice

Exercise Solution

```
PS3="Choose your favorite person:"
select CHOICE in "Josh" "Marty" "Dave" "HAL" "Me"; do
  case $CHOICE in
     "Josh"|"Marty")
            echo "You have chosen wisely."
            ;;
     "Dave"|"HAL")
            echo "2001 is so last decade."
            ;;
     "Me")
            echo "Fine, I see how it is."
            ;;
  esac
  break
done
```

Arrays

```
name=(element1 element2 ... elementN)

name[index]=value          # set an element
$name                      # get first element
${name[index]}             # get an element
${name[*]}                 # elements sep.by spaces
${#name[*]}                # array's length
```

- arrays don't have a fixed length; they can grow as necessary
- if you go out of bounds, shell will silently give you an empty string
  - you don't need to use arrays in assignments in this course

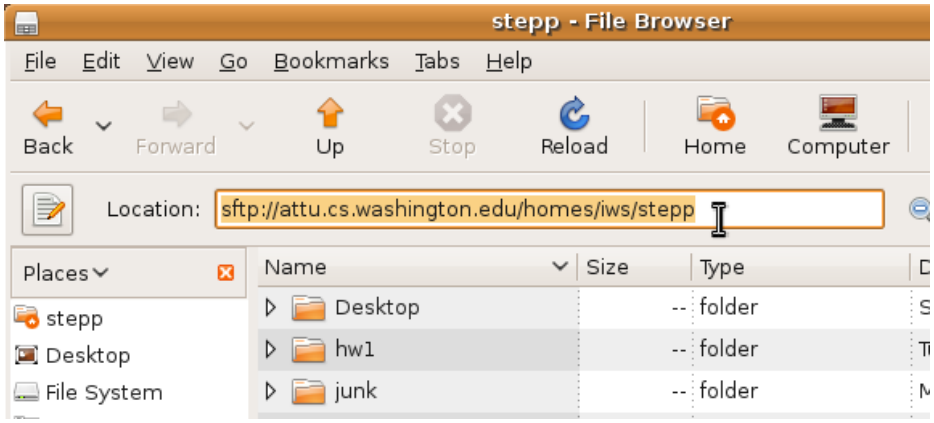
Functions

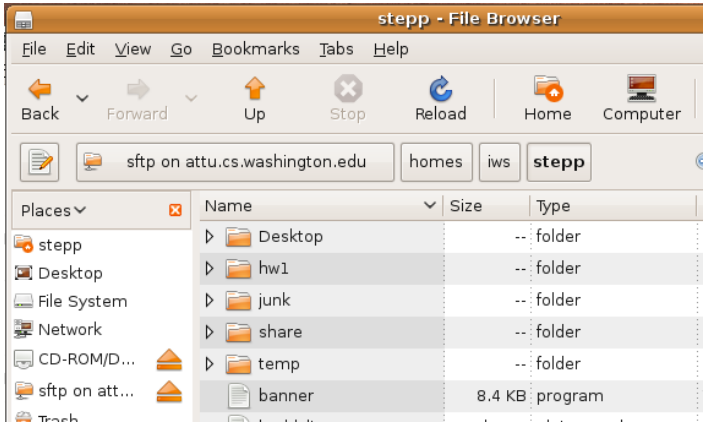
```
function name() {          # declaration
    commands               # ()'s are optional
}

name                       # call
```

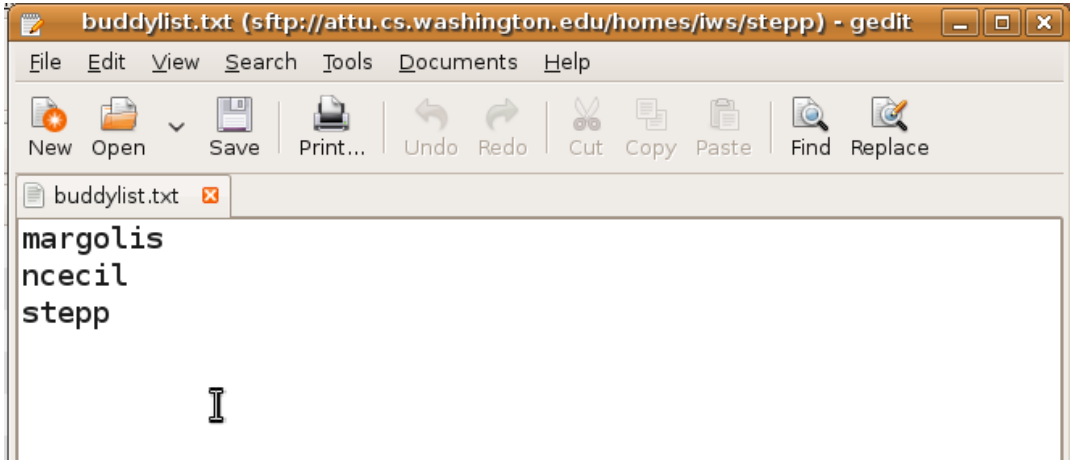
- functions are called simply by writing their name (no parens)
- parameters can be passed and accessed as $1, $2, etc. (icky)
  - you don't need to use functions in assignments in this course

Remote editing

- Gnome's file browser and `gedit` text editor are capable of opening files on a remote server and editing them from your computer
  - press Ctrl-L to type in a network location to open



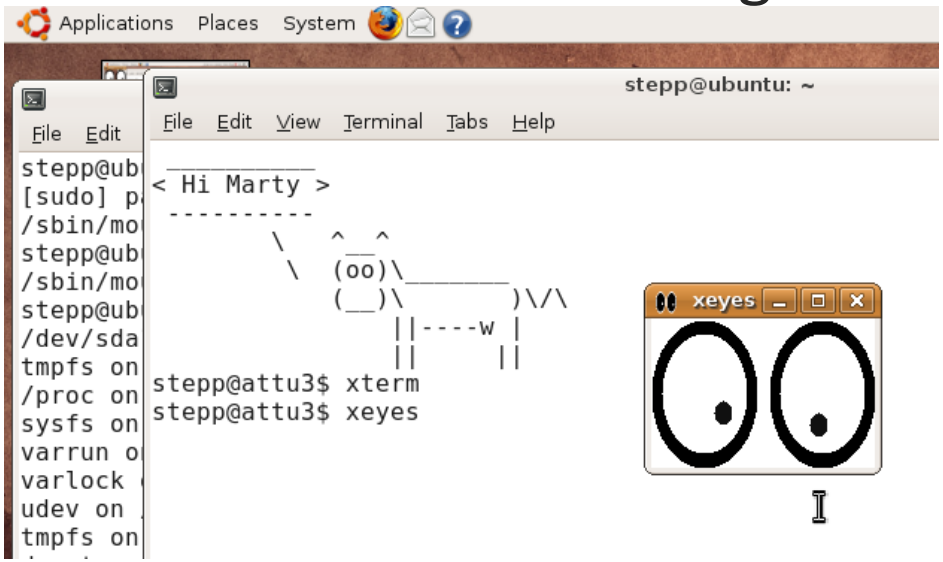




Remote X display

- normally, you cannot run graphical programs on a remote server
- however, if you connect your SSH with the -X parameter, you can!
  - the X-Windows protocol is capable of displaying programs remotely

```
ssh -X attu.cs.washington.edu
```



- Other options (-Y for “Trusted” mode, -C for compressed, see online)

Compressed files

command	description
`zip, unzip`	create or extract .zip compressed archives
`tar`	create or extract .tar archives (combine multiple files)
`gzip, gunzip`	GNU free compression programs (single-file)
`bzip2, bunzip2`	slower, optimized compression program (single-file)

- many Linux programs are distributed as `.tar.gz` archives
  - first, multiple files are grouped into a `.tar` file (not compressed)
  - next, the `.tar` is compressed via `gzip` into a `.tar.gz` or `.tgz`

- to decompress a `.tar.gz` archive:

```
$ tar -xzf filename.tar.gz
```

Other useful tidbits

- Single quotes vs double quotes
  - Quotes tell the shell to treat the enclosed characters as a string
  - Variable names are not expanded in single quotes
    - `STAR=*`
      - `echo $STAR`
      - `echo “$STAR”`
      - `echo ‘$STAR’`
- Shell History
  - The shell remembers all the commands you’ve entered
  - Can access them with the `history` command
  - Can execute the most recent matching command with !
    - Ex: `!less` will search backwards until it finds a command that starts with `less`, and re-execute the entire command line

Newlines in Windows/Unix

- Early printers had two different command characters:
  - Carriage return (\r) – move the print head back to the left margin
  - Line feed (\n) – move the paper to the next line
  - Both occurred when you wanted a "newline"
- As time went on, both (\r\n) and just (\n) were used to signify a "newline"
- Windows typically uses the (\r\n) version, while Unix uses (\n)
  - Can cause problems when displaying text files created on one system on another system
  - Most modern text editors recognize both and do the right thing
  - Can convert if needed:
    - `dos2unix` and `unix2dos` commands