# Introduction to UNIX

CSE 2031

Fall 2012

# Introduction

- UNIX is an operating system (OS).
- Our goals:
  - Learn how to use UNIX OS.
  - Use UNIX tools for developing programs/ software, specifically shell programming.

# Processes

- Each running program on a UNIX system is called a process.
- Processes are identified by a number (process id or PID).
- Each process has a unique PID.
- There are usually several processes running **concurrently** in a UNIX system.

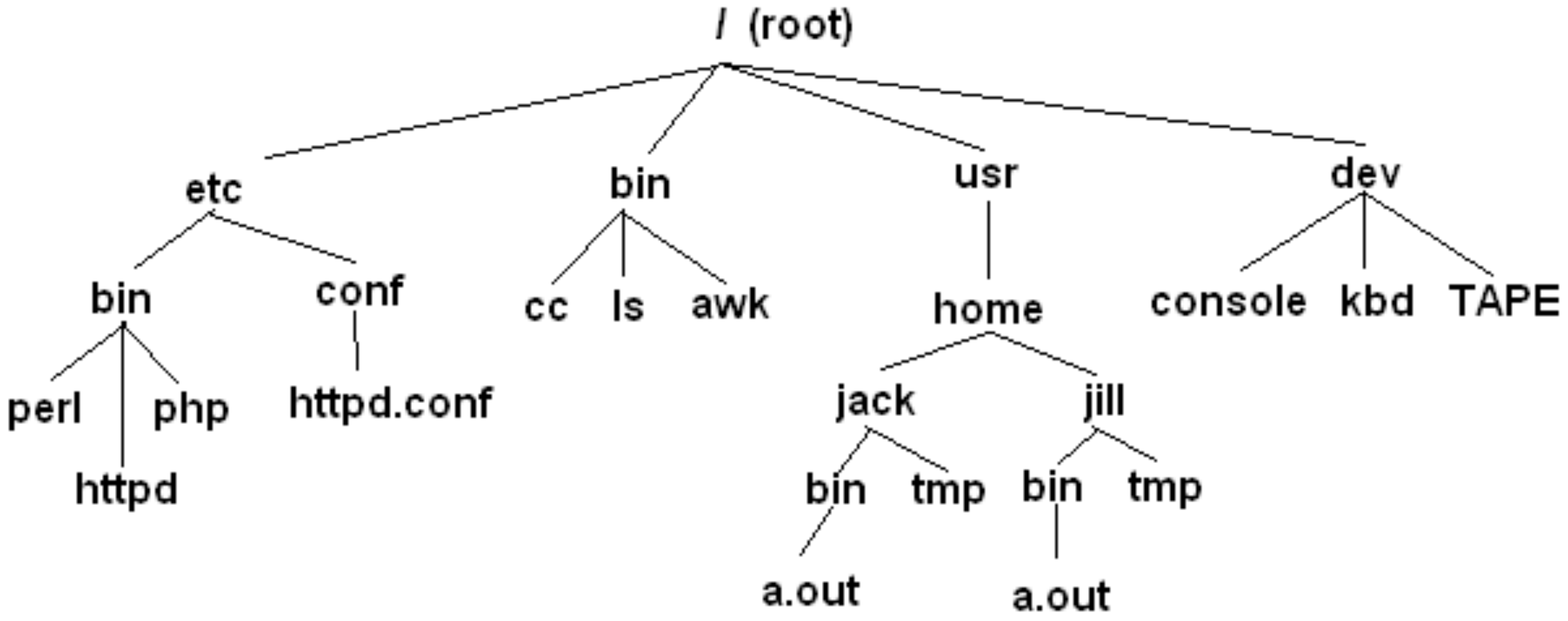
# ps command

```
% ps a      # list all processes
  PID TTY          TIME CMD
 2117 pts/24   00:00:00 pine
 2597 pts/79   00:00:00 ssh
 5134 pts/67   00:00:34 alpine
 7921 pts/62   00:00:01 emacs
13963 pts/24   00:00:00 sleep
13977 pts/93   00:00:00 ps
15190 pts/90   00:00:00 vim
18819 pts/24   00:00:07 stayAlive
24160 pts/44   00:00:01 xterm
. . .
```

# The File System

- Directory structure
- Current working directory
- Path names
- Special notations

# Directory Structure



# Current Working Directory

- Every process has a current working directory.
- In a shell, the command **ls** shows the contents of the current working directory.
- **pwd** shows the current working directory.
- **cd** changes the current working directory to another.

# Path Names

- A path name is a reference to something in the file system.
- A path name specifies the set of directories you have to pass through to find a file.
- Directory names are separated by '/' in UNIX.
- Path names beginning with '/' are absolute path names.
- Path names that do not begin with '/' are relative path names (start search in current working directory).

# Special Characters

- . means the current directory
- .. means the parent directory
  - **cd ..**
  - **cd ../Notes**
- ~ means the home directory
  - **cat ~/lab3.c**
- To go directly to your home directory, type
  - **cd**

# Frequently Used Terminal Keystrokes

- Interrupt the current process: Ctrl-C
- End of file: Ctrl-D
- Read input (stdin) from a file
  - a.out < input_file
- Redirect output (stdout) to a file
  - ls > all_files.txt        # overwrites all_files.txt
- Append stdout to a file
  - ls >> all_files.txt      # append new text to file

# Wildcards (File Name Substitution)

- Goal: referring to several files in one go.
- ? match single character
  - ls ~/C2031/lab5.???
  - lab5.doc lab5.pdf lab5.out
- * match any number of characters
  - ls ~/C2031/lab5.*
- […] match any character in the list enclosed by [ ]
  - ls ~/C2031/lab[567].c
  - lab5.c lab6.c lab7.c
- We can combine different wildcards.
  - ls [ef]*.c
  - enum.c ex1.c fn2.c

# Unix Commands

There are many of them

We will see some of the most useful ones

We know already:

ls, cp, mv, rm, pwd, mkdir, rmdir, man

# cat, more, tail

```
% cat phone_book
Yvonne 416-987-6543
Amy 416-123-4567
William 905-888-1234
John 647-999-4321
Annie 905-555-9876
```

```
% more phone_book
```

Similar to cat, except that the file is displayed one screen at a time.

```
% tail myfile.txt
```

Display the last 10 lines

```
% tail -5 myfile.txt
```

Display the last 5 lines

```
% tail -1 myfile.txt
```

Display the last line

```
% tail +3 myfile.txt
```

Display the file starting from the 3rd line.

# echo

- When one or more strings are provided as arguments, echo by default repeats those strings on the screen.

```
% echo This is a test.
This is a test.
```

- It is not necessary to surround the strings with quotes, as it does not affect what is written on the screen.
- If quotes (either single or double) are used, they are not repeated on the screen.

```
% echo ‘This is’”a test.”
This is a test.
```

- To display single/double quotes, use `\’` or `\”`

# echo (cont.)

```
% echo a \t b
a t b
% echo 'a \t b'
a        b
% echo "a \t b"
a        b
```

# wc

```
% wc enum.c
 14  37 220 enum.c

% wc [e]*.c
 14  37 220 enum.c
 17  28 233 ex1.c
 21  46 300 ex2.c
 52 111 753 total
```

```
% wc -c enum.c
220 enum.c

% wc -w enum.c
37 enum.c

% wc -l enum.c
14 enum.c
```

# sort

```
% cat phone_book
Yvonne 416-987-6543
Amy 416-123-4567
William 905-888-1234
John 647-999-4321
Annie 905-555-9876

% sort phone_book
Amy 416-123-4567
Annie 905-555-9876
John 647-999-4321
William 905-888-1234
Yvonne 416-987-6543
```

Try these options:

**sort -r**

reverse normal order

**sort -n**

numeric order

**sort -nr**

reverse numeric order

**sort -f**

case insensitive

# cmp, diff

```
% cat phone_book
Yvonne 416-987-6543
Amy 416-123-4567
William 905-888-1234
John 647-999-4321
Annie 905-555-9876

% cat phone_book2
Yvonne 416-987-6543
Amy 416-111-1111
William 905-888-1234
John 647-999-9999
Annie 905-555-9876
```

```
% cmp phone_book phone_book2
phone_book phone_book2
differ: char 9, line 2

% diff phone_book
phone_book2
2c2
< Amy 416-123-4567
---
> Amy 416-111-1111
4c4
< John 647-999-4321
---
> John 647-999-9999
```

# who

```
% who
ossama   pts/13       Nov  7 00:22 (ip-198-96-36-11.dynamic.yorku.ca)
hoda      pts/21       Nov  4 16:49 (gomez.cs.yorku.ca)
gordon   pts/24       Nov  5 10:40 (bas2-toronto08-1096793138.dsl.bell.ca)
minas    pts/29       Nov  2 14:09 (monster.cs.yorku.ca)
jas       pts/37       Oct 18 12:36 (brayden.cs.yorku.ca)
utn      pts/93       Nov  7 12:21 (bas2-toronto44-1177753778.dsl.bell.ca)
```

- User name
- Terminal associated with the process
- Time when they logged in

# kill

```
% ps a
  PID TTY          TIME CMD
 2117 pts/24   00:00:00 pine
 2597 pts/79   00:00:00 ssh
 5134 pts/67   00:00:34 alpine
 7921 pts/62   00:00:01 emacs
13963 pts/24   00:00:00 sleep
13976 pts/43   00:00:00 sleep
13977 pts/93   00:00:00 ps
15190 pts/90   00:00:00 vim
24160 pts/44   00:00:01 xterm
. . .
```

```
% kill -9 7921
```

9 is the KILL signal

# history

```
% history 10
   323  12:45   ls
   324  12:47   cd Demo_2031/
   325  12:48   ls
   326  12:48   m ex1.c
   327  12:49   who
   328  12:50   history 10
   329  12:52   ls -a
   330  12:56   ls Stack/
   331  12:57   ls
   332  12:57   history 10
```

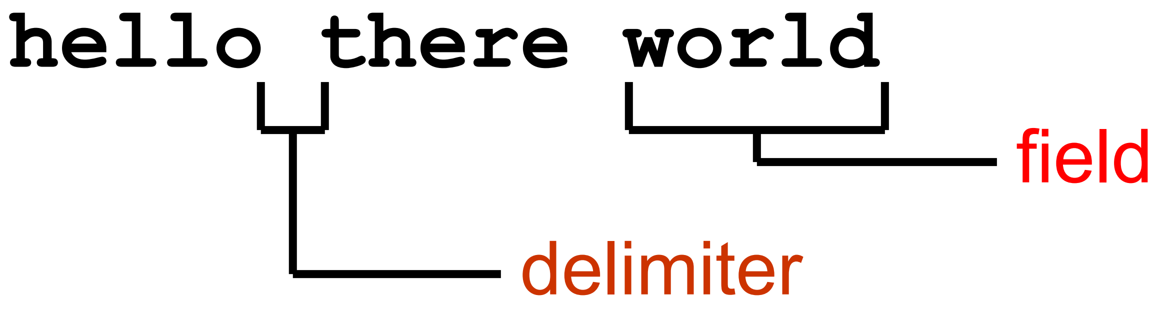
# Pipes

- Pipe: a way to connect the output of one program to the input of another program without any temporary file.
- Pipeline: connection of two or more programs through pipes.
- Examples:

```
ls -l | wc -l          # count number of files
who | sort             # sort user list
who | wc -l            # count number of users
```

# cut

- Used to split lines of a file
- A line is split into fields
- Fields are separated by delimiters
- A common case where a delimiter is a space:



```
hello there world
```

field

delimiter

# cut

- Syntax
- **`cut [-f`***`fields`***`] [-c`***`columns`***`]`**
  **`[-d`***`character`***`] [`***`filename`*** **`…]`**
- If filenames are given on command line, input is taken from those files
- If no filenames are given, input comes from stdin
- This approach to input is very common

# cut

- Two main forms - extracting fields
- `cut -f3 -d,`
  - extract field 3 from each line
  - fields are separated by ',’

- e.g. with an input of
- `hello,there,world,!`
- output would be just "world"

# cut

- The other way - pulling out characters:
- `cut -c30-40`
  - extract characters 30 through 40 (inclusive) from each line
- Note that we can use ranges (e.g. 4-10) or lists (e.g. 4,6,7) as values for -f or -c.

# uniq

- Removes repeated lines in a file
- `uniq [-c] [input [output]]`

- Notice difference in args:
  - 1st filename is input file
  - 2nd filename is output file
- If input is not specified, use stdin
- If output is not specified, use stdout

# uniq

- Only works for lines that are adjacent, e.g.
- **`abacus`**
- **`abacus`**
- **`bottle`**
- **`abacus`**
- becomes
- **`abacus`**
- **`bottle`**
- **`abacus`**

# uniq

- With the **`-c`** option output is a count of how many times each line was repeated
- For previous input:

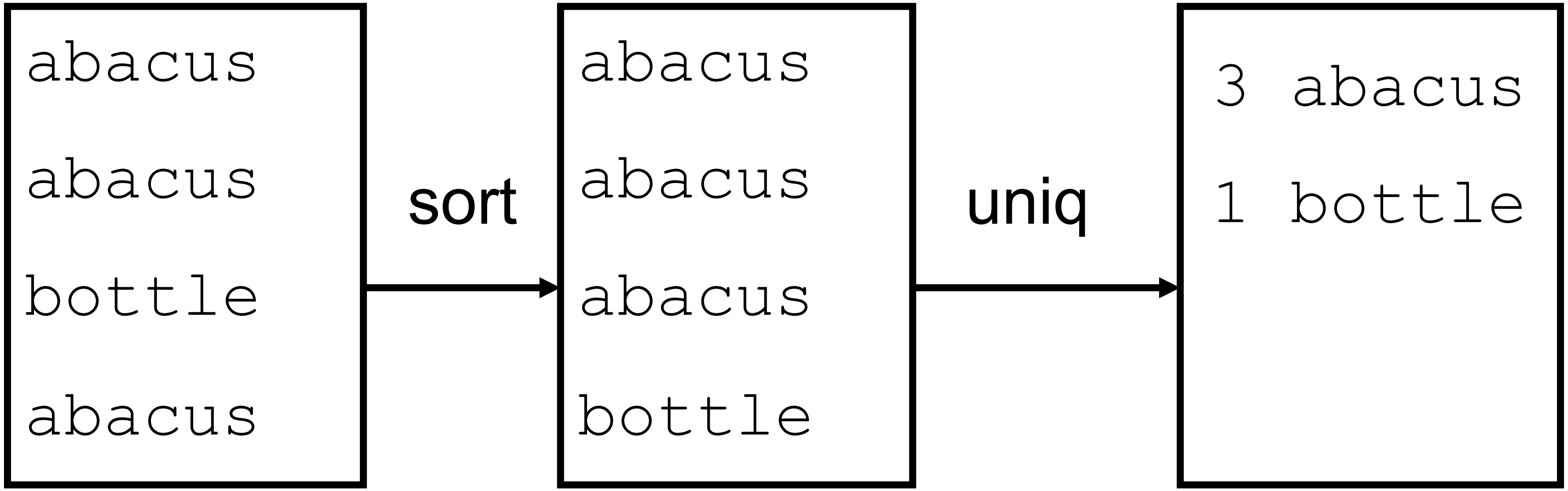
```
   2 abacus
   1 bottle
   1 abacus
```

# sort + uniq

- uniq is a little limited but we can combine it with sort
- `sort | uniq -c`
- counts number of times line appears in file
- output would now be:
- `3 abacus`
- `1 bottle`

# sort + uniq

- To understand:



```
abacus
abacus
bottle
abacus
```

sort

```
abacus
abacus
abacus
bottle
```

uniq

```
3 abacus
1 bottle
```

# tr

- "translates" characters
- Maps characters from one value to another
- **tr** *string1 string2*
- **tr [-d] [-c]** *string*
- Input is always stdin, output is always stdout
- A character in string1 is changed to the corresponding character in string2

# tr

- A simple example:

-
  ```
  tr x y
  ```
- All instances of 'x' are replaced with 'y'
- Each string can be a set of characters

-
  ```
  tr ab xy
  ```
- 'a' is replaced with 'x', 'b' is replaced with 'y'

# tr

- The -d option means delete the given characters
- `tr -d xyz`
- Delete all 'x', 'y', and 'z' characters
- The -c option means "complement" (i.e. the inverse)
- `tr -d -c xyz`
- Delete all characters except 'x', 'y', and 'z'

# Why Are These So Weird?

- Unix philosophy:
  Do one thing and do it well
- So 'tr' doesn't know how to read from files, the "cat" command does know how:

-
  ```
  cat filename | tr …
  ```

# Regular Expressions

- A regular expression is a special string (like a wildcard pattern)
- A compact way of matching several lines with a single string

# Regular Expressions

- The basics:
- letters and numbers are literal - that is they match themselves:
- e.g. **`"foobar"`** matches **`"foobar"`**
- **`'.'`** matches any character (just one)
- e.g. **`"fooba."`** matches **`"foobar"`**, **`"foobat"`**, etc.

# Regular Expressions

- Each `'.'` character must match exactly one character
- e.g. "**`f..bar`**" matches "**`foobar`**" but not "**`fubar`**"
- **`[xxx]`** matches any character in the set
- e.g. "**`foob[aeiou]r`**" matches "**`foobar`**", "**`foober`**", "**`foobir`**", etc.

# Regular Expressions

- `'*'` means "0 or more of the last character"
- `"fo*"` matches `"f"`, `"fo"`, `"foo"`, `"fooo"`, `"foooo"`, etc.
- `"[0-9][0-9]*"` matches a decimal number
- `".*"` matches anything (including an empty string)
- `'?'` means "0 or 1 of the last character"

# Regular Expressions

- **"^"** matches the beginning of the line, **"$"** matches the end of the line
- **"^foobar"** - matches any line that starts with "foobar"
- **"foobar$"** - matches any line that ends with "foobar"

# grep

- Prints out all lines in the input that match the given regular expression
- **grep [***options***]** *pattern* **[***file* …**]**
- e.g.
- `grep hello`
- Prints out all lines containing "hello"

# grep

- A warning:  does the following work?
- `grep ^[a-z]*`
- If you type it in, it won't work
- Why not?

# grep

- Options control searches:
- **`-i`** - case-insensitive search (don't distinguish between 'a' and 'A')
- **`-v`** - invert search (print out lines which don't match)
- **`-l`** - when used with filenames, print out names of files with matching lines

# grep

- Some interesting uses:
- `grep -v '^#'`
- Removes all lines beginning with '#'
- `grep -v '^[ ]*$'`
- Removes all lines which are either empty or contain only spaces

# fgrep

- Like grep, fgrep searches for things but does not do regular expressions - just fixed strings
- fgrep == faster grep
- **`fgrep 'hello.*goodbye'`**
- Searches for string “hello.*goodbye” - does not match it as a regular expression

# Working With Files

- Wildcards are limited
- The following commands helps us to find files and run commands on them

# find

- Finds files with the given properties
- **`find`** *`path`* `…` **`[-`***`operation`* **`…]`**
- Not just regular files - includes directories, devices - everything it finds in the filesystem
- Starts at the given path and walks down through every directory it finds

# find

- We can specify operators to control
  - which files we find
  - what to do with them when we find them
- All operators begin with "-", e.g.
- `find $HOME -print`
- Prints out the name of every file in your home directory

# find

- Operators are handled left-to-right
- Each operator is "true" or "false"
- Stop processing operators for a file if an operator is false
- e.g. **`"-print"`** means print out the file name and is always "true"

# find

- Another operator: **`-type`** *`filetype`*
- Tests to see what kind of file it is
- e.g. f = regular file, d = directory
- **`find $HOME -type d -print`**
  - Prints all directories under your home directory.

# find

- **`-name`** *`pattern`* = true if the name of the file matches the wildcard pattern 'pattern’
- **`find $HOME -type f -name '*.c’`**

Finds all files under your home directory which are regular files and end in “.c”

- So what can you do with this?
  - look at '-exec' operator for find!

# xargs

- Another way to use find is to combine it with xargs
- **`xargs`** *`command`*
  - xargs executes given command for each word in its stdin

    ```
    find $HOME -type f -name '*.c′
    -print | xargs wc -l
    ```
- Counts number of words in all C files

# NEVER-DO List in UNIX

- Never switch off the power on a UNIX computer.
  - You could interrupt the system while it is writing to the disk drive and destroy your disk.
  - Other users might be using the system.
- Avoid using **`*`** with **`rm`** such as **`rm *, rm *.c`**
- Do not name an important program core.
  - When a program crashes, UNIX dumps the entire kernel image to a file called **`core`**.
  - Many scripts go around deleting these **`core`** files.
- Do not name an executable file **`test`**.
  - There is a Unix command called **`test`**.

# Command Terminators

- Command terminator: new line or **;**

```
% date; who
```

- Another command terminator: &

```
% nedit lab9.c&
```

  - Tells the shell not to wait for the command to complete.
  - Used for a long-running command “in the background” while you continue to use the xterm for other commands.

# Command Terminators (cont.)

- Use parentheses to group commands

```
% ( sleep 5; date ) & date
14929        # process ID of long-running command
Tue Nov 9 14:06:15 EST 2010   # output of 2nd date
% Tue Nov 9 14:06:20 EST 2010 # output of 1st date
```

- The precedence of | is higher than that of ;

```
% date; who | wc -l
% (date; who) | wc -l
```

# tee command

- **tee** copies its input to a file as well as to standard output (or to a pipe).

```
% date | tee date.out
Tue Nov  9 13:51:22 EST 2010
% cat date.out
Tue Nov  9 13:51:22 EST 2010
% date | tee date.out | wc
      1        6      29
% cat date.out
Tue Nov  9 13:52:49 EST 2010
```

# Comments

- If a shell word begins with #, the rest of the line is ignored.
- Similar to // in Java.

```
% echo Hello #world
Hello
% echo Hello#world
Hello#world
```

# Metacharacters

- Most commonly used: *
- Search the current directory for file names in which any strings occurs in the position of *

```
% echo *  # same effect as
% ls *
```

- To protect metacharacters from being interpreted: enclose them in single quotes.

```
% echo '***'
***
```

# Metacharacters (cont.)

- Or to put a backslash \ in front of each character:

```
% echo \*\*\*
***
```

- Double quotes can also be used to protect metacharacters, but …
- The shell will interpret `$`, `\` and `…` inside the double quotes.
- So don’t use double quotes unless you intend some processing of the quoted string (see slide 10).

# Quotes

- Quotes do not have to surround the whole argument.

```
% echo x'*'y    # same as echo 'x*y'
x*y
```

- What's the difference between these two commands?

```
% ls  x*y
% ls 'x*y'
```

# Program Output as Arguments

- To use the output of a command X as the argument of another command Y, enclose X in back quotes: **`X`**

```
% echo `date`
Tue Nov 9 13:11:03 EST 2010
% date     # same effect as above
Tue Nov 9 13:11:15 EST 2010
% echo date
date
% wc `ls *`
% wc *     # same as above
```

# Program Output as Arguments (2)

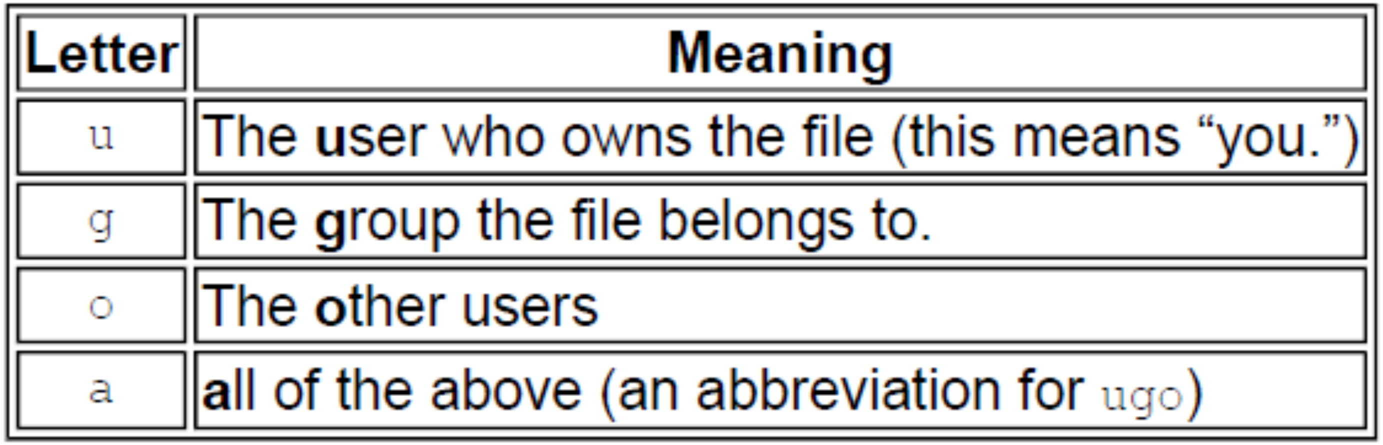
- Single quotes vs. double quotes:

```
% echo The time now is `date`
The time now is Tue Nov 9 13:11:03 EST 2010

% echo "The time now is `date`"
The time now is Tue Nov  9 13:11:15 EST 2010

% echo 'The time now is `date`'
The time now is `date`
```

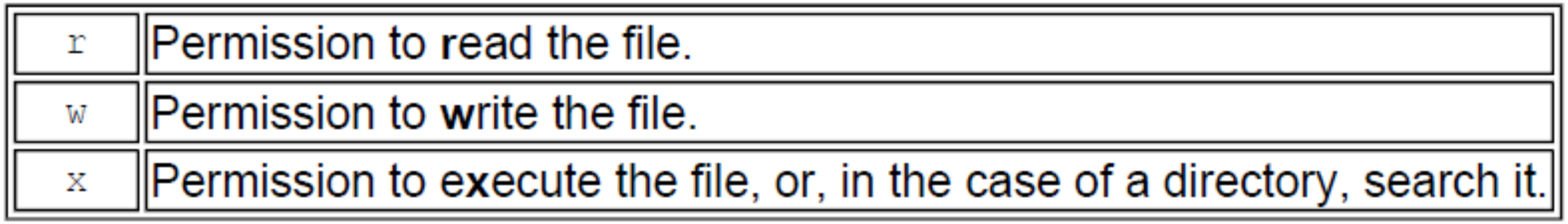
# Program Output as Arguments (3)

```
% pwd
/cs/home

% ls -1 | wc -l
26

% echo You have `ls -1 | wc -l` files in the `pwd` directory
You have 26 files in the /cs/home directory
```

# File/Directory Permissions



| Letter | Meaning |
|---|---|
| u | The **u**ser who owns the file (this means "you.") |
| g | The **g**roup the file belongs to. |
| o | The **o**ther users |
| a | **a**ll of the above (an abbreviation for ugo) |



| | |
|---|---|
| r | Permission to **r**ead the file. |
| w | Permission to **w**rite the file. |
| x | Permission to e**x**ecute the file, or, in the case of a directory, search it. |

# chmod Command

```
chmod who+permissions filename # or dirname
chmod who-permissions filename # or dirname
```

Examples:

```
chmod u+x my_script # make file executable
chmod a+r index.html      # for web pages
chmod a+rx Notes          # for web pages
chmod a-rx Notes
chmod a-r index.html
```

# chmod with Binary Numbers

```
chmod u+x my_script    chmod 700 my_script
chmod a+r index.html   chmod 644 index.html

chmod a+rx Notes       chmod 755 Notes
chmod a-rx Notes       chmod 700 Notes
                       chmod 750 Notes
chmod a-r index.html   chmod 600 index.html
                       chmod 640 index.html
```

# chgrp Command

```
chgrp grp_name filename  # or dirname
```

- Examples:

```
chgrp submit asg1
chgrp labtest lab9
```

- To display the group(s) a user belongs to, use **id** command:

```
% id cse12345
uid=12695(cse12345) gid=10000(ugrad) groups=10000(ugrad)
```

# Next time …

- Writing Shell Scripts

- Reading: Chapters 1, 2, 3.1 – 3.5
  “Practical Programming in the UNIX Environment”

- **`chmod`** tutorial:

  **`http://catcode.com/teachmod/`**