- Question 1.C, 1.D
- Question 5
- Quiz
- Preview of HW3
- Introduction to Unix

Quiz

- Write an MIPS assembly program that computes the factorial of a given input. The integer input is passed through register $a0, and the result is returned in register $v0.
- Show the contents of the stack after each function calls, assuming that the input is 4.

Why Bother?

- Most programmers who learn UNIX end up finding it useful
- Provides powerful command-line interface
  - Many simple tasks are *easier* to accomplish
  - Possible to script repetitive operations
- Widely used in research and industry, and runs most of the servers on the Internet

UNIX Philosophy

- Multiuser / multitasking
- Toolbox approach
  - Combine multiple simple commands instead of using a single complex application
- Designed by programmers for programmers

Shelling into CSUG

- From Windows, use PuTTY

  http://www.chiark.greenend.org.uk/~sgtatham/putty/

  (Google can give you this URL)

  – Demo

- From MacOS, open a terminal and type
  - **`ssh netid@csug01.csuglab.cornell.edu`**

Command Line Environment

- Shell is the command line interpreter
  - Just another program
  - Bourne shell (**`bash`**)
  - C Shell (**`csh`**)
- Default shell in CSUG is **`tcsh`**
- This talk uses bash
  - Switch to bash: **`exec bash`**

Running Commands

- Commands follow the form:
  - **`command <options> <arguments>`**
  - Options modify the command
  - Arguments indicate what file to operate on
- Get help by typing **`man command`**
- Example:

```
[msiegen@tiger ~]$ ls -l /usr
total 301
drwxr-xr-x    2 root root 69632 Oct 18 08:43 bin/
drwxr-xr-x    2 root root  4096 Aug 12  2004 etc/
drwxr-xr-x    2 root root  4096 Aug 12  2004 games/
drwxr-xr-x  117 root root 20480 Sep 12 20:40 include/
...
```

Plumbing

- I/O Redirection
  - `>` Redirect standard output to file
  - `>>` Append standard output to file
  - `<` Get input from file
- Pipes (|) are used to take the output of one program and use it as input to another

  e.g. **`du -sk /home/* | sort -nr | head -10 > disk_hogs.txt`**

Practical Tips

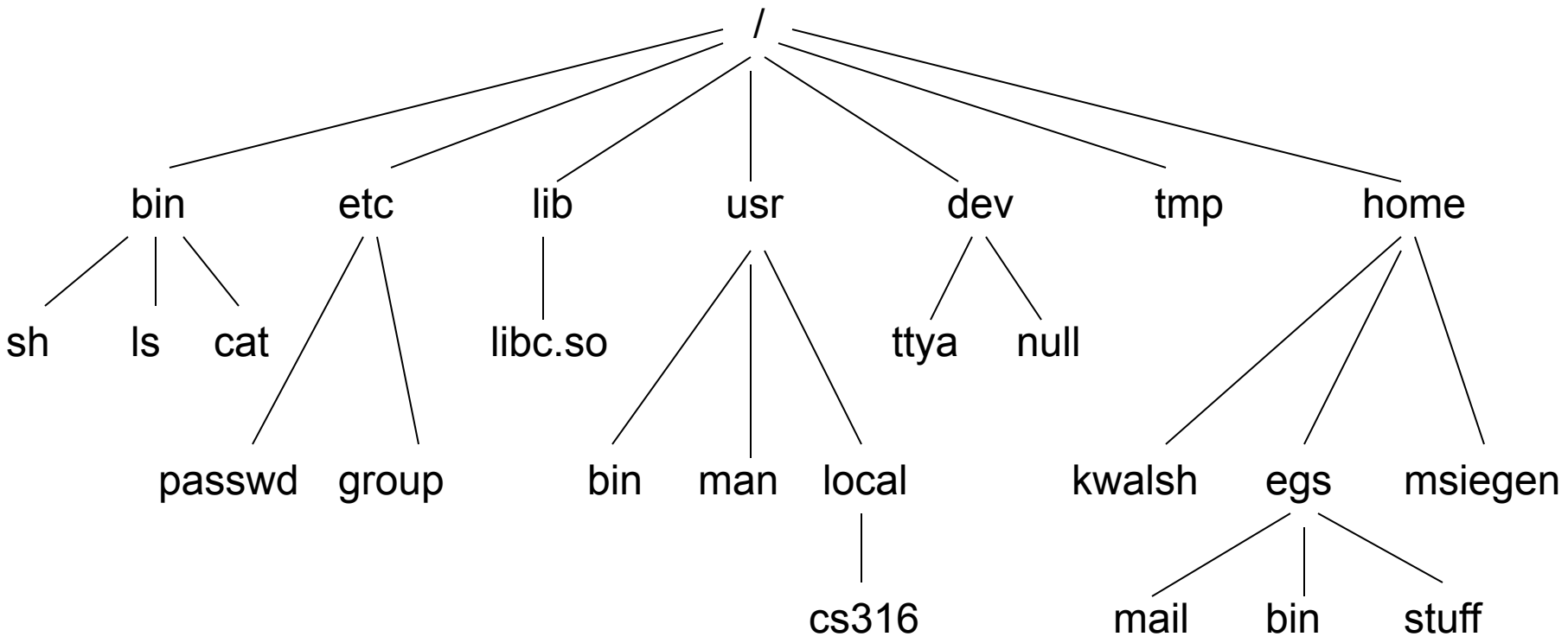
- Use **`less`** to view output that will not fit on your screen

  e.g. **`ls -lR | less`**
- Use grep to filter output, and wc to count lines

  e.g. **`ps aux | grep “vim” | wc -l`**
- Use **`&&`** to run multiple commands in sequence

  e.g. **`./configure && make && make install`**
- Many more possibilities!

File System


/
bin
etc
lib
usr
dev
tmp
home
sh
ls
cat
passwd
group
libc.so
bin
man
local
cs316
ttya
null
kwalsh
egs
msiegen
mail
bin
stuff

File System

- Case sensitive!
- Moving around, working with directories

`cd`	Change working directory
`pwd`	Print working directory
`ls -la`	List all files in working directory
`mkdir`	Make directory
`rmdir`	Remove directory
`cp`	Copy file
`mv`	Move or rename file
`rm`	Delete a file

- Searching

  e.g. **`find -name Makefile`**

Setting Up for HW1

- Copy the HW1 files into your home directory:

```
cp -R /usr/local/cs316/hw1_codeExamples ~
```

- Fix your path:

```
export PATH=\
$PATH:/usr/local/cs316/mipsel-linux/bin
```

- Demo compiling **`hw1.s`** and **`hw2.s`**

Viewing File Contents

- Use **`cat`** or **`less`**:

```
$ cat hw1.c  # use cat for short files
#include “test-include.h”

_start() {
}

$ less hw1.s  # use less for long files
```

Comparing Files

- Use **`diff`**:

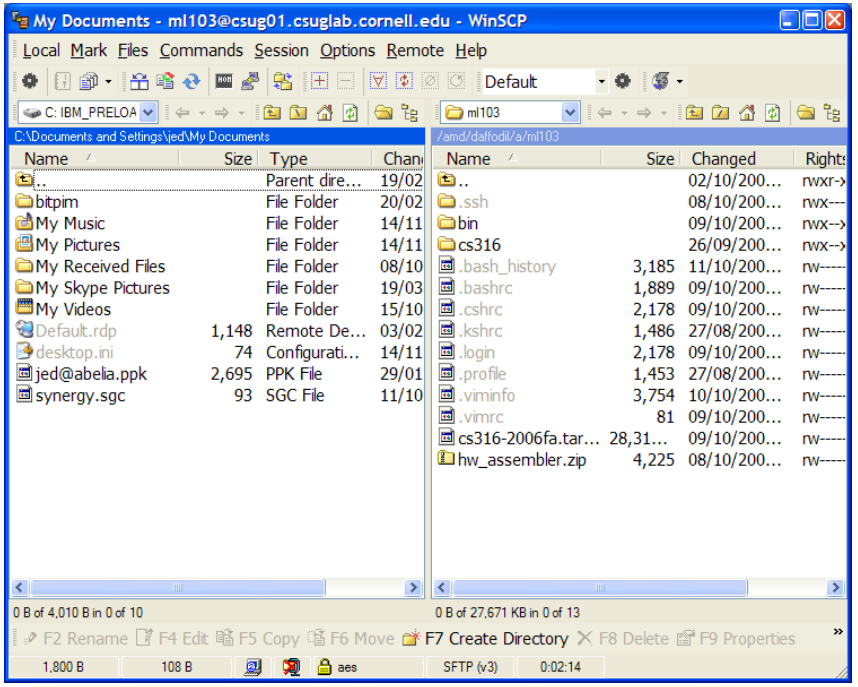
```
$ cat file1
Hello!
This is the contents of file1.
Goodbye.
$ cat file2
Hello!
This is the contents of file2.
Goodbye.
$ diff -u file1 file2
--- file1       2007-10-11 04:25:28.000000000 -0400
+++ file2       2007-10-11 04:25:45.000000000 -0400
@@ -1,3 +1,3 @@
Hello!
-This is the contents of file1.
+This is the contents of file2.
Goodbye.
```

- Demo: **`diff -u hw1.s hw2.s`**

Transferring Files

- Use WinSCP

  http://winscp.net/



Further Reading

- Manual (man) pages
- O'Reilly Books
  - Free access on campus:

    http://proquest.safaribooksonline.com/
  - Or from home through the Safari Tech Books link at:

    http://www.englib.cornell.edu/erg/shortlist.php

Plumbing

- Running multiple commands in sequence
  - Use semicolon (;) to run commands unconditionally
  - Use double ampersand (&&) to run commands only until the first error occurs
- Use parentheses to group a sequence and redirect output

  e.g. **`(date && ls) > logfile`**

  Not the same as: **`date && ls > logfile`**

Wildcards

- Shorthand for referencing multiple existing files on the command line
  - * any number of characters
  - ? exactly one character
  - [abc] any one of a, b, or c
  - [!abc] any character except a, b, or c
- Examples

```
ls -l *.c
lpr [Mm]akefile
```

File System Permissions

- Permissions can be specified for
  - Owner
  - Group
  - All
- Permissions are
  - Read
  - Write
  - Execute
- Example:

```
-rwxr-xr-x  1 msiegen ta   10152 Sep 21 17:04 disassemble
-rw-r-----  1 msiegen ta     329 Sep 21 17:04 main.c
```

  The disassembler may be executed by anyone on the system, but the source file may only be read by people in the **`ta`** group. Both files may only be edited by the user **`msiegen`**.

File System Permissions

- For a directory, “read” means being able to list its contents, “execute” means being able to access files within the directory
  - Unless the files have more restrictive permissions
- Use chmod to add or remove permissions (rwx) for user, group, and others (ugo):

  `chmod ugo+x` Let anyone execute

  `chmod go-w` Prevent non-owner form writing
- Or, specify absolute permissions in octal
  - 4=r, 2=w, 1=x
  - e.g. 755=rwxr-xr-x, 640=rw-r-----

    e.g. **`chmod 755 filename`**

Job Control

- Use & after a command to place job in background
- Manage jobs:
  - jobs — List jobs
  - fg %1 — Bring job 1 to foreground
  - bg %2 — Run job 2 in background
  - kill %3 — Terminate job 3
  - ^Z — (control+Z) suspend foreground job
  - ^C — (control+C) terminate foreground job

Job Control

- Example

```
[msiegen@tiger ~]$ sleep 800 &
[1] 16139
[msiegen@tiger ~]$ sleep 400 &
[2] 16141
[msiegen@tiger ~]$ jobs
[1]-  Running                 sleep 800 &
[2]+  Running                 sleep 400 &
[msiegen@tiger ~]$ kill %1
[msiegen@tiger ~]$ jobs
[1]-  Terminated              sleep 800
[2]+  Running                 sleep 400 &
[msiegen@tiger ~]$ fg %2
sleep 400
 ^Z
[2]+  Stopped                 sleep 400
[msiegen@tiger ~]$ bg %2
[2]+ sleep 400 &
```

Environment Variables

- Display all variables by typing **`env`**
- Set a variable, example:

  `NETID=abc123; export NETID` (bourne shell)

  `setenv NETID abc123` (c-shell)
- Use a variable in a command, example:

  `echo $NETID`
- Environment variables are used to control various behaviours of the shell, as well as pass global information to other programs that are started from within the shell
- The variable $PATH is used to locate programs that are run

Beyond a Single User

`ps aux` List all running processes

`who`; **`w`** Show who else is logged in

`top` Show CPU, memory usage

(useful for finding out why a system is *soooo* slow, and who to blame)

Some Useful Commands

- file Determine the type of a file
- sort Sort lines in a text stream
- uniq Eliminate duplicate lines
- wc Count bytes, words, or lines
- cal Display a calendar
- grep Filter a text stream
- sed Search and replace on text stream
- awk (Slightly) more advanced scripting

Advanced Topics

- Shell scripting
  - Anything which can be done from the command line, can be scripted
- Regular expressions
  - Fancier version of wildcards
  - Allows complex matching and search and replace operations on text
  - Suppored by grep, awk, and many scripting/ programming languages