CSC207 Software Design Lectures

Using Unix

What It Is

An operating system
- Invented in the early 1970s
- Has been copied and modified many times
- Best-known variants today are Linux and Solaris

A set of tools
- Most of which now run on other operating systems
- If you're using Windows at home, you will use the Cygwin package in this course

A way of thinking about programming
- Most Unix commands read and write text
- Can easily be connected together to solve simple problems
- Examples given below

Terminology

The *kernel* is a program that:
- Runs other programs
- Manages the file system
- Handles input and output devices

A *shell* is a program that gives a user an interactive command prompt
- Reads commands from the keyboard
- Asks the kernel to run programs
- Displays the output from those programs

Many different shells can (and do) run on top of the same kernel
- Most commonly used are `tcsh` (your default) and `bash`

Logging On

In the lab:
- Find an unused machine
- Type in your user name and password
- Should see a set of windows, some of which are interactive shells

From home, use SSH (Secure SHell) to get an interactive shell

See "Work Remotely" link on CDF "Working at Home" page

At home, use Cygwin

Instructions are on this CDF page

or this guide was written by a CSC207 student in the summer of 2003

What's Here?

Type `ls` at a shell prompt

That's a lower-case 'L', not a '1'

`ls` stands for `list`

Shows you the files in the current directory

```
$ ls
CVS                setup-at-home.html      using-cvs.html
index.html         unix-commands.html
```

Creating Directories

Type `mkdir demo`

Again, without the quotation marks

"mkdir" is the command

"make directory"

Yes, Unix is often cryptic

`demo` is the *argument* to the command

Just like an argument to a function

Tells `mkdir` what directory to make

Doesn't print anything

Moving Around

Type `cd demo`

"change directory" to `demo`

As far as the shell is concerned, you are now "in" that directory

Technically, it is your *current working directory*

Type `pwd` ("print working directory")

Shows what directory you're in

```
$ mkdir demo
$ cd demo
```

```
$ pwd
/u/gvwilson/demo
```

Paths

An *absolute path* is one spelled out in full from the top (or *root*) of the file system

A *relative path* is one spelled out from where you presently are

Two special symbols in paths:

- "." means "current working directory"
- ".." means "parent of this directory"

So try this:

```
$ pwd
/u/gvwilson/demo
$ mkdir nested
$ cd nested
$ ls
$ ls ..
nested
```

Removing Directories

Use `rmdir`

Only works if the directory is empty

Yes, you *can* delete the directory you're in

```
$ cd ..
$ ls
nested
$ rmdir nested
$ ls
```

Editing Files

`pico` is the simplest editor around

So type `pico a.txt` to edit a new file `a.txt`

Other popular editors are:

- `nedit`
- `vi`
- `emacs`

But I strongly recommend that you use Eclipse or JBuilder for programming

Copying Files

Use the `cp` command to copy files

`cp file1 file2` copies `file1` to `file2`

Overwrites `file2` if it already exists

`cp file1 dir` copies `file1` into the directory `dir`, creating `dir/file1`

```
$ pwd
/u/gvwilson/demo
$ pico a.txt
$ cp a.txt b.txt
$ ls
a.txt   b.txt
$ mkdir nested
$ cp a.txt nested
$ ls
a.txt   b.txt
$ ls nested
a.txt
$ cp a.txt nested/c.txt
$ ls nested
a.txt   c.txt
```

Deleting Files

Use `rm` (for "remove")

```
$ pwd
/u/gvwilson/demo
$ cd nested
$ ls
a.txt   c.txt
$ rm *.txt
$ ls
$ cd ..
$ rmdir nested
$ ls
a.txt   b.txt
```

The expression `*.txt` means "everything ending in .txt"

The `*` is a *wildcard*

What do you think `rm a*.txt` does?

Or `rm a.*`?

Or `rm *`?

Note: wildcards work with *all* commands

Shell does pattern matching

Hands results to the command

Other Commands

Start with:

- `man`: shows manual pages
- `diff`: shows differences between files
- `passwd`: use it to change your password
- `wc`: count characters, words, and lines in a file

This page lists other important commands

You are expected to learn them for this course

- I.e., they are examinable material

Redirection

Almost all Unix programs read from ***standard input*** and write to ***standard output***

- Standard input ("stdin") is usually the keyboard
- Standard output ("stdout") is usually the screen

You can ***redirect*** stdin using `<`

- Tells the shell to get the command's input from a file

Similarly, redirect stdout using `>`

- Tells the shell to send the command's output to a file

```
$ pwd
/u/gvwilson/demo
$ ls
a.txt   b.txt
$ cat a.txt
This is a text file.
$ cat a.txt > copy.txt
$ ls
a.txt   b.txt   copy.txt
$ cat copy.txt
This is a text file.
$ wc a.txt
       1       5      21 a.txt
$ wc a.txt > count.txt
$ ls
a.txt   b.txt   copy.txt   count.txt
$ cat count.txt
       1       5      21 a.txt
$ ls > files
$ cat files
a.txt
b.txt
copy.txt
count.txt
```

```
files
$ wc < files
       5       5      37
$ rm b.txt copy.txt count.txt files
$ ls
a.txt
```

End of File and Interrupt

How do you signal end of input when a command is running interactively?

On Unix, use `^D` (control-D, not "^" followed by "D")

```
$ wc
This
is
a
test
^D
       4       4      15
```

Note: may need `^Z` on Windows instead of `^D`

Note also: when the shell sees end of input, it logs you out

- So if you type `^D` to the prompt, your session ends

Use `^C` (control-C) to interrupt a running program

History

The `history` command will show you your most recent commands

- But *not* their output

You can repeat a command by typing `!123`, where `123` is the command number

```
$ ls
a.txt
$ history
...
1043 ls
1044 history
$ !1043
ls
a.txt
```

Combining Commands

Can connect the output of one command to the input of another using `|`

- Called a *pipe*

`abc | xyz` is the same as:

```
abc > tmp.txt
xyz < tmp.txt
rm tmp.txt
```

```
$ pwd
/u/gvwilson/demo
$ ls
a.txt
$ ls | wc
       1       1       6
$ cat > b.txt
This
file
contains
repeated
words
Some
words
are
repeated
in
this
file
^D
$ sort b.txt
This
file
contains
repeated
words
Some
words
are
repeated
in
this
file
$ sort b.txt | uniq -c
      1 are
      1 contains
      2 file
      1 in
      2 repeated
      1 Some
      1 this
      1 This
      2 words
$ sort b.txt | uniq -c | grep 2 | sort
      2 file
      2 repeated
      2 words
```

Scripts

You can store shell commands in a file, and then execute it

Called a *shell script*

```
$ cat > example.sh
echo "example.sh is running"
ls
wc *.txt
^D
$ sh example.sh
example.sh is running
a.txt  b.txt  example.sh
      1       5      21 a.txt
     12      12      71 b.txt
     13      17      92 total
```

Note: this means that the shell can be programmed

Environment Variables

Shell stores values in *environment variables*

- Just like variables in a programming language

Syntax for setting values depends on your shell

- For tcsh: `setenv COURSE csc207`
- For bash: `export COURSE=csc207`

Refer to variables as `$VARIABLE_NAME`

- E.g. `echo $COURSE` will print `csc207`

```
$ setenv CMD example.sh
$ echo $CMD
example.sh
$ sh $CMD
example.sh is running
a.txt  b.txt  example.sh
      1       5      21 a.txt
     12      12      71 b.txt
     13      17      92 total
```

Shell automatically sets some environment variables for you

- `EDITOR`: your default editor
- `HOME`: your home directory
- `HOSTNAME`: the computer you are on

Login Scripts

Whenever a shell starts, it executes a special script stored in your home directory

For tcsh: `$HOME/.cshrc`

For bash: `$HOME/.bashrc`

Use this to set up environment variables that you want every time you log in

You will need to set several for the exercises in this course

Musings

A lot of people get religious about operating systems

Remember: it's just another tool

And also remember: tools shape their users, too

$Id: unix.html,v 1.1 2003/09/06 20:41:52 gvwilson Exp $