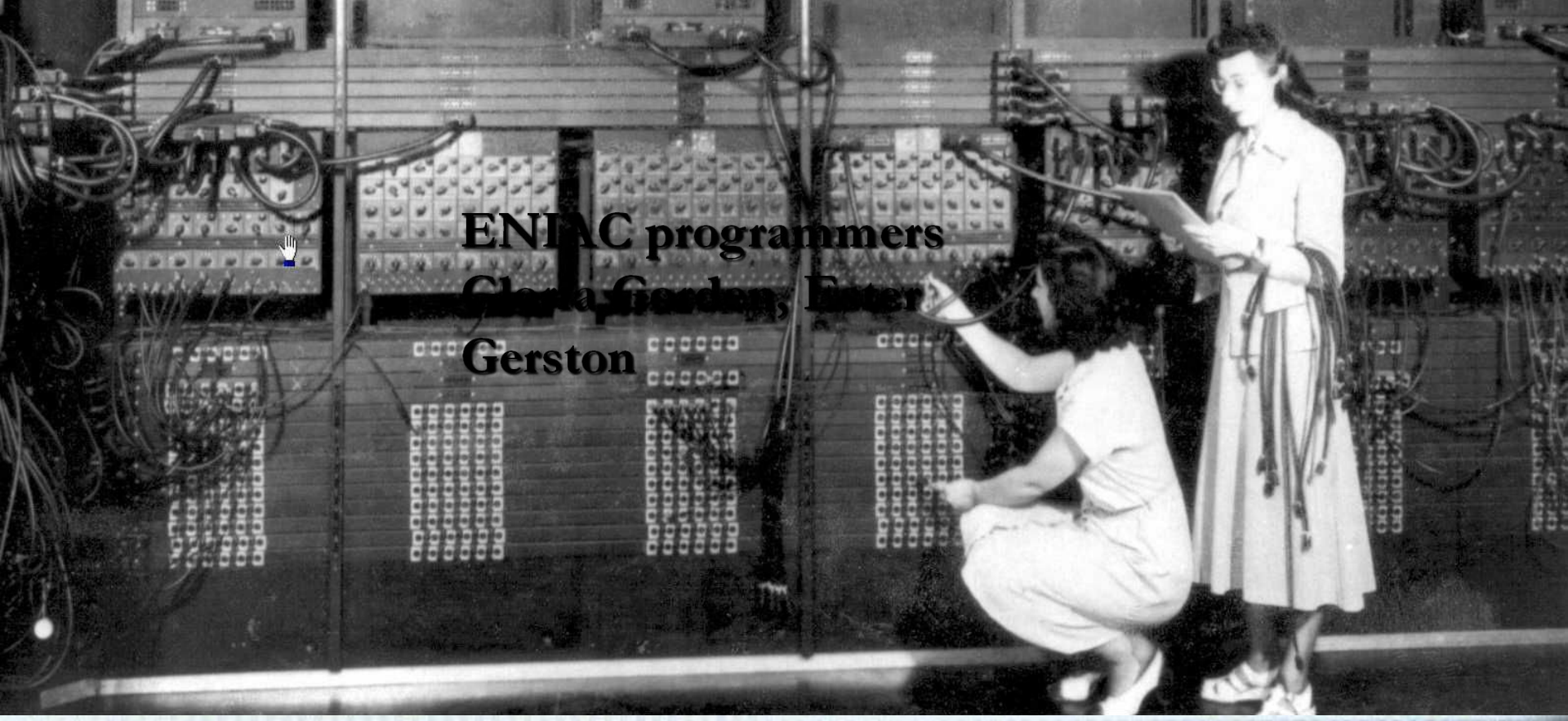
计算物理

Lecture 2

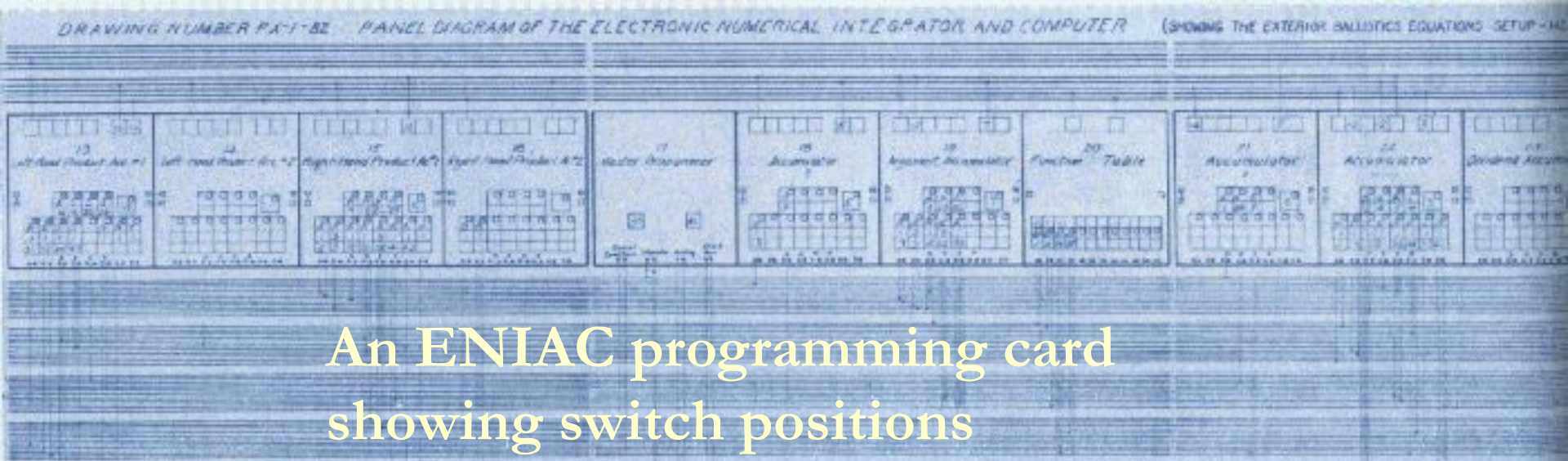
傅子文

Outline of Lecture 2

- Methods – getting started with Unix shells
  - Notes editted from Greg Wilson's "Software Carpentry" course
    - http://www.swc.scipy.org/– this is a FANTASTIC resource
- 10,000 ft view of programming
- You can skip this lecture if
  - You know what a shell is
  - You know the difference between an absolute path and a relative path
  - You know what **ls**,**cp**, and **wc** do
  - You understand redirection and pipes



ENIAC programmers
Gerston



An ENIAC programming card showing switch positions

A note on software standards

- Experimental results are only publishable if they are believed to be ***correct*** and ***reproducible***
  - Equipment calibrated, samples uncontaminated, relevant steps recorded, etc.
- In practice, rely on expectations and cultural norms
  - Drilled into people starting with their first high school chemistry class
  - Only actually check work that is already under suspicion
- How well do computational scientists meet these standards?
- Correctness of code rarely questioned
  - We all know programs are buggy…
  - …but when was the last time you saw a paper rejected because of concerns over software quality?
- Reproducibility often nonexistent
  - How many people can reproduce, much less trace, each computational result in their thesis?

Industry isn't a whole lot better

- Commercial projects of all sizes routinely go over time and over budget
- What they deliver is often incomplete, riddled with bugs, and not what the customer actually wanted
- How is this possible?
- Low expectations
  - Like American cars in the 1970s
- Lack of accountability
  - Hard to sue software developers
  - Most shrink-wrap licenses effectively say, "This CD could be blank, and we wouldn't have to give you back your money."

Solutions are available though

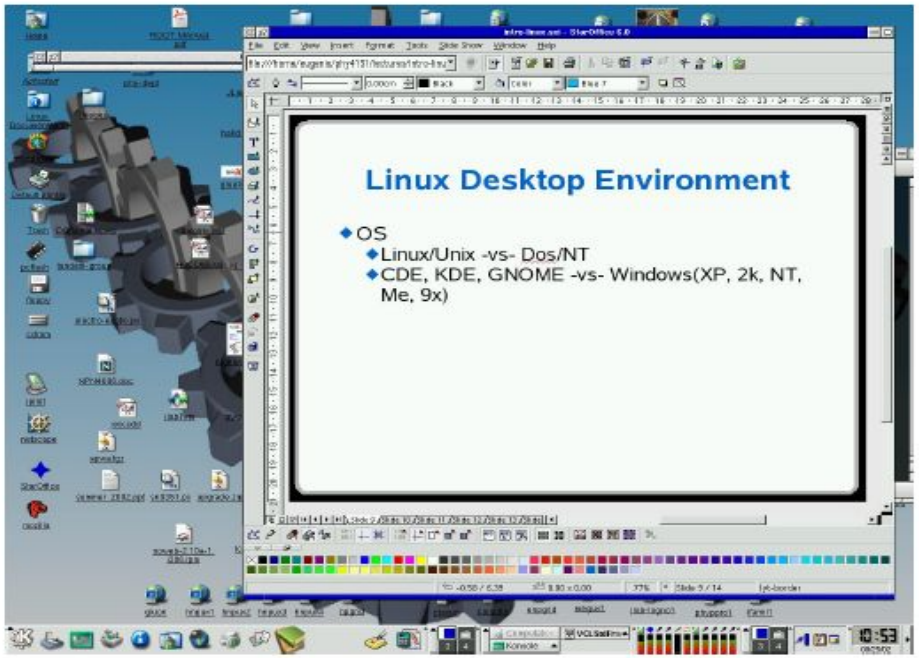
- …and we've known about them for years
  - They just aren't evenly distributed
- This is one of the reasons good programmers are up to "28 times" better than bad ones
  - See "Facts & Fallacies of Software Engineering" by Robert L Glass
- *Improving quality improves productivity*
  - The more effort you put into making sure it's right the first time, the less total time it'll take to get it built
- *The tools and techniques that help you write better code also help you write more code, faster*
  - Version control (such as CVS, RCS)
  - Symbolic debuggers (e.g. DBX, see the primer)
  - Test-driven development (developing standard test cases that must be passed)

Linux Desktop Environment

- OS
  - Linux/Unix ↔ DOS/Windows NT

ubuntu

- Desktop->Development Environments
  - KDE, GNOME ↔ Windows 10, macOS, iOS, Android



CLUI *vs* GUI

- Most modern tools have a graphical user interface (GUI)
  - They're easier to use
- But command-line user interfaces (CLUIs) still have their place
  - Easier to build a simple CLUI than a simple GUI
  - Higher action-to-keystroke ratio
    - Once you're over the learning curve
- Easier to see and understand what the computer is doing on your behalf
  - Which is part of what this course is about
- Most important: it's easier to combine CLUI tools than GUI tools
  - Small tools, combined in many ways, can be very powerful
- This lecture focuses on Unix
  - Because while there are good Unix emulators for Windows, there aren't good Windows emulators for Unix

The Unix Shell

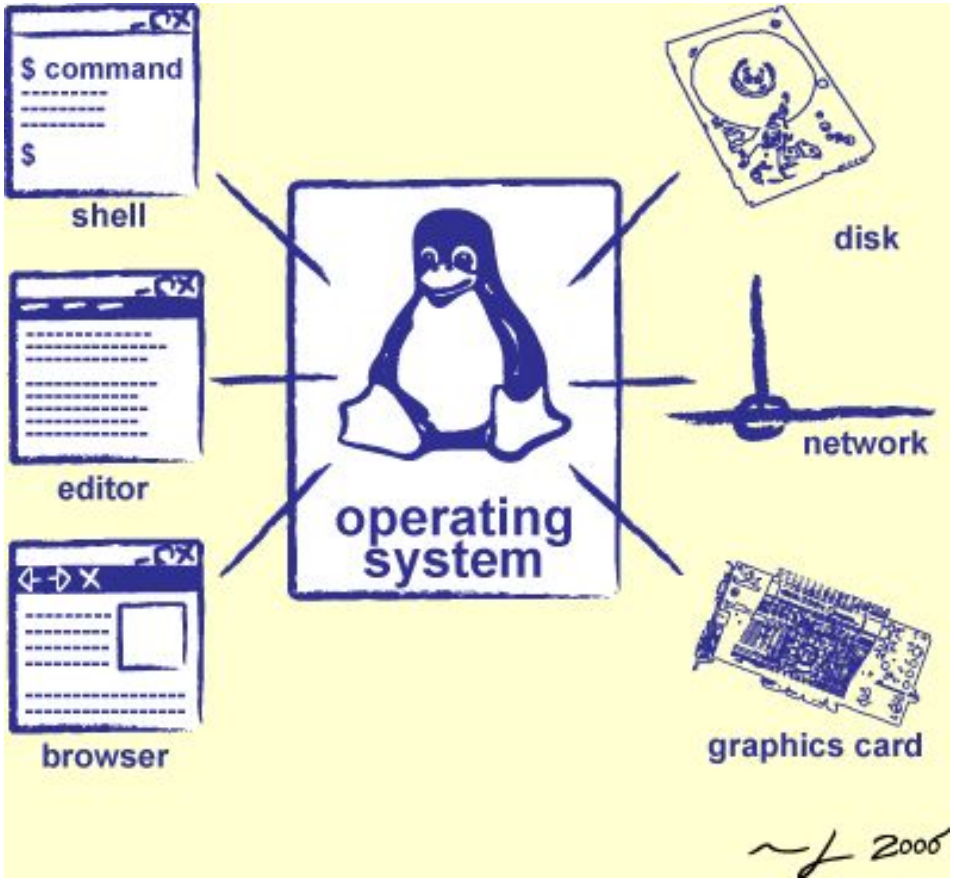
~/swc/trunk

```
Greg Wilson@marandine ~/swc/trunk
$ ls
LICENSE.txt   config.mk  docs       img         license.swc  scraps   swc.dtd  util
Makefile      data        extern     index.swc   press        sites    tests    web
conf          depend.mk   graphics   lec         print.css    swc.css  tmp

Greg Wilson@marandine ~/swc/trunk
$ ls -l
total 64
-rwxr-xr-x  1 Greg Wilson None  1070 Dec  8 15:40 LICENSE.txt
-rwxr-xr-x  1 Greg Wilson None 17030 Jan 19 16:34 Makefile
drwxr-xr-x+ 3 Greg Wilson None     0 Jan 25 13:40 conf
-rwxr-xr-x  1 Greg Wilson None  1807 Feb  1 12:38 config.mk
drwxr-xr-x+ 6 Greg Wilson None     0 Dec  9 16:49 data
-rw-r--r--  1 Greg Wilson None  8598 Feb  1 12:39 depend.mk
drwxr-xr-x+ 5 Greg Wilson None     0 Jan  3 09:53 docs
drwxr-xr-x+ 3 Greg Wilson None     0 Jan 27 09:38 extern
drwxr-xr-x+ 7 Greg Wilson None     0 Feb  1 13:02 graphics
drwxr-xr-x+ 3 Greg Wilson None     0 Jan  3 09:53 img
-rwxr-xr-x  1 Greg Wilson None  2912 Jan  3 09:53 index.swc
drwxr-xr-x+ 6 Greg Wilson None     0 Feb  1 12:40 lec
-rwxr-xr-x  1 Greg Wilson None  1302 Dec  8 15:41 license.swc
drwxr-xr-x+ 7 Greg Wilson None     0 Dec  9 16:49 press
-rwxr-xr-x  1 Greg Wilson None   926 Dec  8 15:40 print.css
drwxr-xr-x+ 3 Greg Wilson None     0 Feb  1 08:58 scraps
drwxr-xr-x+ 4 Greg Wilson None     0 Dec  8 15:40 sites
-rwxr-xr-x  1 Greg Wilson None  6950 Jan 16 14:42 swc.css
-rwxr-xr-x  1 Greg Wilson None  1037 Jan 11 13:17 swc.dtd
drwxr-xr-x+ 5 Greg Wilson None     0 Jan  5 10:52 tests
drwxr-xr-x+ 3 Greg Wilson None     0 Feb  1 12:57 tmp
drwxr-xr-x+ 3 Greg Wilson None     0 Feb  1 12:57 util
drwxr-xr-x+ 4 Greg Wilson None     0 Feb  1 13:00 web

Greg Wilson@marandine ~/swc/trunk
$ _
```

- The most important command-line tool is the *command shell*
- Usually just called "the shell"
- Looks (and works) like an interactive terminal circa 1980
- Many different shells have been written
- The Bourne shell, called sh, is an ancestor of many of them
- We'll use bash (the Bourne Again Shell) or csh (c shell)

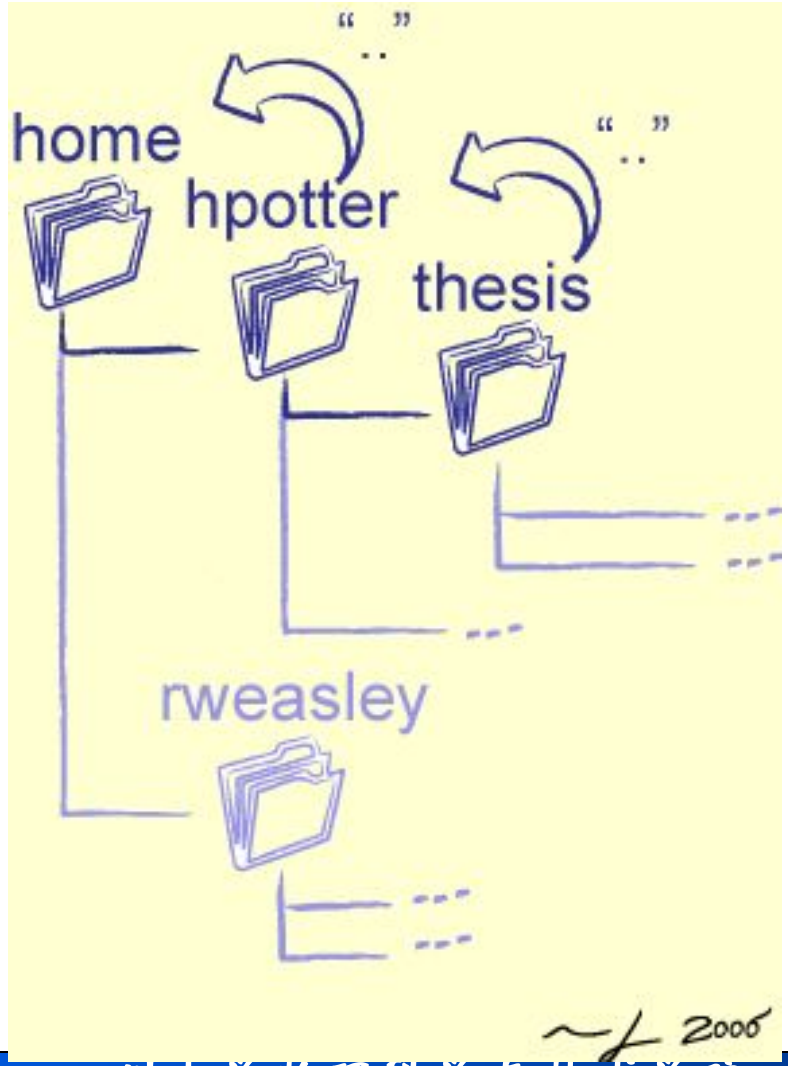
The Shell is not the Unix OS

- The *operating system* is *not* just another program
  - Automatically loaded when the computer boots up
  - Runs everything else (including shells)
- The OS manages the computer's hardware
  - Provides a common interface to different chips, disks, network cards, etc.
  - So that user-level applications can run anywhere
  - The OS also keeps track of what programs are running, what privileges they have, etc.
  - Which makes it crucial to security



Filesystem

- Files are stored in directories (often called folders)
  - Results in the familiar *directory tree*
  - Remember - items in different directories can have the same name
- On Unix, the file system has a unique *root directory* called /
  - Every other directory is a child of it, or a child of a child, etc.
- On Windows, every *drive* has its own root directory
  - So C:\home\Admin\notes.txt is different from J:\home\Admin\notes.txt
  - When you're using Cygwin, you can also write C:\home\Admin as c:/home/Admin
  - Or as /cygdrive/c/home/Admin
    - Some Unix programs give ":" a special meaning, so Cygwin needed a way to write paths without it…



First steps

- Easiest way to learn basic Unix commands is to see them in action
- Type **pwd** (short for "print working directory") to find out where you are
  - Unfortunately, most Unix commands have equally cryptic names

```
$ pwd
/home/Admin
```

- Then type **ls** (for "listing") to see what's in the current directory

```
$ ls
data  hello.dat  hello.dat~
```

- To see what's in the data directory, type **ls data**

```
$ ls data
file.txt    listing.dat
```

Getting around the shell

- Or: type **cd data** to "go into" data
  -i.e., change the current working directory to data
- Type ls on its own
- Type cd .. to go back to where you started

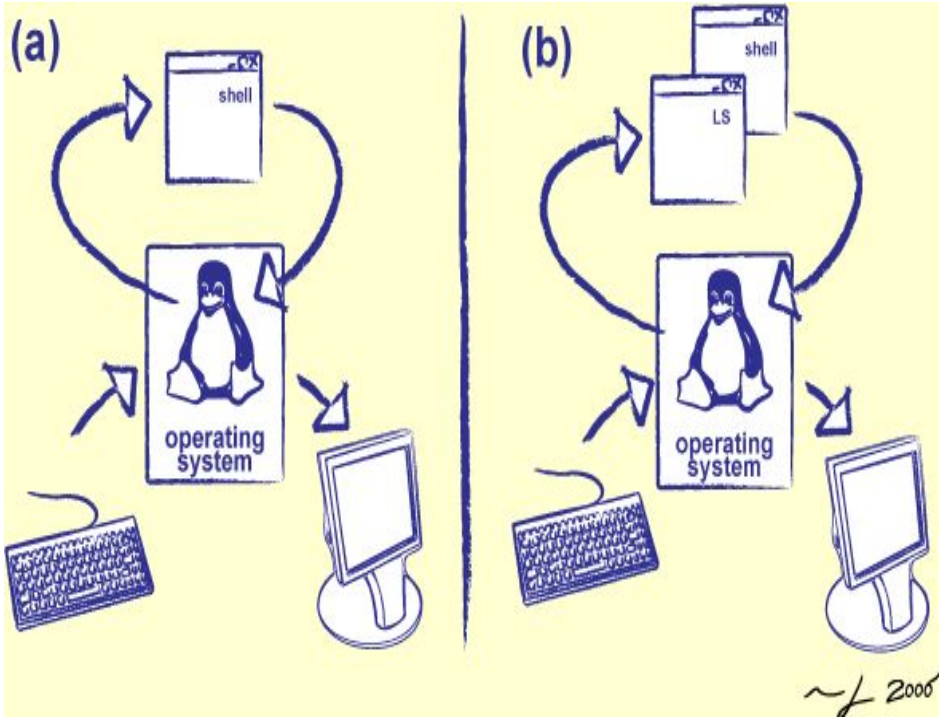
```
$ cd data
$ pwd
/home/Admin/data
$ ls
file.txt  listing.dat
$ cd ..
$ pwd
/home/Admin/
```

Paths

- A *path* is a description of how to find something in a file system
- An *absolute path* describes a location from the root directory down
  - Equivalent to a street address
  - Always starts with "/"
  - /home/Admin is Admin's home directory
- A *relative path* describes how to find something from some other location
  - Equivalent to saying, “Four blocks north, and seven east”
  - From /home/Admin, the relative path to file.txt is /data/file.txt
- Every program (including the shell) has a *current working directory*

Execution cycle

- When you type a command like **ls**, the OS:
  - Reads characters from the keyboard
  - Passes them to the shell (because it's the currently active window)
- The shell:
  - Breaks the line of text it receives into words
  - Looks for a program with the same name as the first word
    - See in a moment how the shell knows where to look
  - Runs that program
- That program's output goes back to the shell…
  - …which gives it to the OS…
  - …which displays it on the screen
- All well-designed software systems work this way
  - Break the task down into pieces
  - Write a tool that solves each sub-problem



Unix Manual

- You can find out information about any command, *e.g.* **ls** in this case, by typing

```
$man ls
```

- The resulting page will tell you all about the command
  - May seem a dense and difficult at first, but after a while you get used to the format and things become quite obvious

Providing options to commands

- Can make ls produce more informative output by giving it some *flags*
  - By convention, flags for Unix tools start with "-", as in "**-c**" or "**-l**"
  - Some flags take arguments (such as filenames)
- Show directories with trailing slash

```
$ ls -F
data/  hello.dat  hello.dat~
```

- Show all files and directories, including those whose names begin with .
  - By default, ls doesn't show things whose names begin with .
    - So that . and .. don't always show up

```
$ ls –a
.   .bash_history  .bashrc   .inputrc   data      hello.dat~
..  .bash_profile  .emacs.d  .inputrc~  hello.dat
```

Creating Files & Directories

- Rather than messing with the course files, let's create a temporary directory and play around in there

```
$ mkdir tmp
```

  - Note: no output (but **-v** ("verbose") would tell mkdir to print a confirmation message)

- Go into that directory: no files there yet

```
$ cd tmp
$ ls
```

- Use the editor of your choice to create a file called earth.txt with the following contents:

```
Name: Earth
Period: 365.26 days
Inclination: 0.00
Eccentricity: 0.02
```

A note on editors

- On a windows machine you can always use notepad
- The standard Unix editor is "**vi**"
  - Non-trivial to use, has both a command and editting mode
  - Good to know though, since you are pretty much guaranteed to have it on any system
- "**emacs**" is the most popular editor
  - A bit easier to use, quite powerful
- "**nano**" is stripped down versions of emacs that are very easy to use
  - Used in the mail program **pine** for composing messages

I recommend using nano if you haven't used an editor before,
Become familiar with those and then learn what you need for vi and emacs

Rapid editting

- Easiest way to create a similar file venus.txt is to copy earth.txt and edit it:

```
$ cp earth.txt venus.txt
$ nano venus.txt
$ ls -t
venus.txt earth.txt
```

  - **-t** tells **ls** to list by modification time, instead of alphabetically

Looking at files

- Check the contents of the file using **cat** (short for "concatenate")
  - Prints the contents of a file to the screen

```
$ cat venus.txt
Name: Venus
Period: 224.70 days
Inclination: 3.39
Eccentricity: 0.01
```

- Compare the sizes of the two files using **ls –l**

```
$ ls -l
total 2
-rw-r--r-- 1 Admin None 69 Oct  2 11:29 earth.txt
-rw-r--r-- 1 Admin None 69 Oct  2 11:34 venus.txt
```

  - Fifth column is size in bytes
- We can also get details about the number of words and characters

wc – word count

```
$ wc earth.txt venus.txt
 4  9  73 earth.txt
 4  9  73 venus.txt
 8 18 146 total
```

– Columns show lines, words, and characters

File ownership & permissions

- On Unix, every user belongs to one or more groups
  - The **groups** command will show you which ones you are in
- Every file is owned by a particular user *and* a particular group
  - Can assign read (r), write (w), and execute (x) permissions independently to user, group, and others
    - Read: can look at contents, but not modify them
    - Write: can modify contents
    - Execute: can run the file (e.g., it's a program)
- **ls -l** shows this information
  - Along with the file's size and a few other things
- Permissions displayed as three rwx triples
- "Missing" permissions shown by "-"
- So rw-rw-r-- means:
  - User and group can read and write
  - Everyone else can read, but not write
  - No one can execute

File & directory permissions

- Change permissions using **chmod** (u=user, g=group, o=world)
  - **chmod u+x hello** allows hello's owner to run it
  - **chmod o-r notes.txt** takes away the world's read permission for notes.txt
- Any set of shell commands can be turned into a program!
  - If it's worth doing again, it's worth automating
- Create a file called nojunk with the following commands

```
#!/usr/bin/bash
rm -f *.junk
```

  - Use man ls to find out what the "-f" flag does

More on permissions

- #!/usr/bin/bash means “run this using the Bash shell”
  - Any program name can follow the #!
  - We'll see some possibilities later
- Change permissions to rwxr-xr-x
- Run it with **./nojunk**
- Don't call your temporary test programs **test**
  - There's already another program called test
- You’ve just turned a file into an executable script (or shell script)

Useful commands

man	Documentation for commands.
cat	Concatenate and display text files.
cd	Change working directory.
chmod	Change permissions
clear	Clear the screen.
cp	Copy files and directories.
date	Display the current date and time.
diff	Show differences between two text files.
echo	Print arguments.
head	Display the first few lines of a file.
ls	List files and directories.
mkdir	Make directories.
more	Page through a text file.
mv	Move (rename) files and directories.
od	Display the bytes in a file.
passwd	Change your password.
pwd	Print current working directory.
rm	Remove files.
rmdir	Remove directories.
sort	Sort lines.
tail	Display the last few lines of a file.
uniq	Remove adjacent duplicate lines.
wc	Count lines, words, and characters in a file.

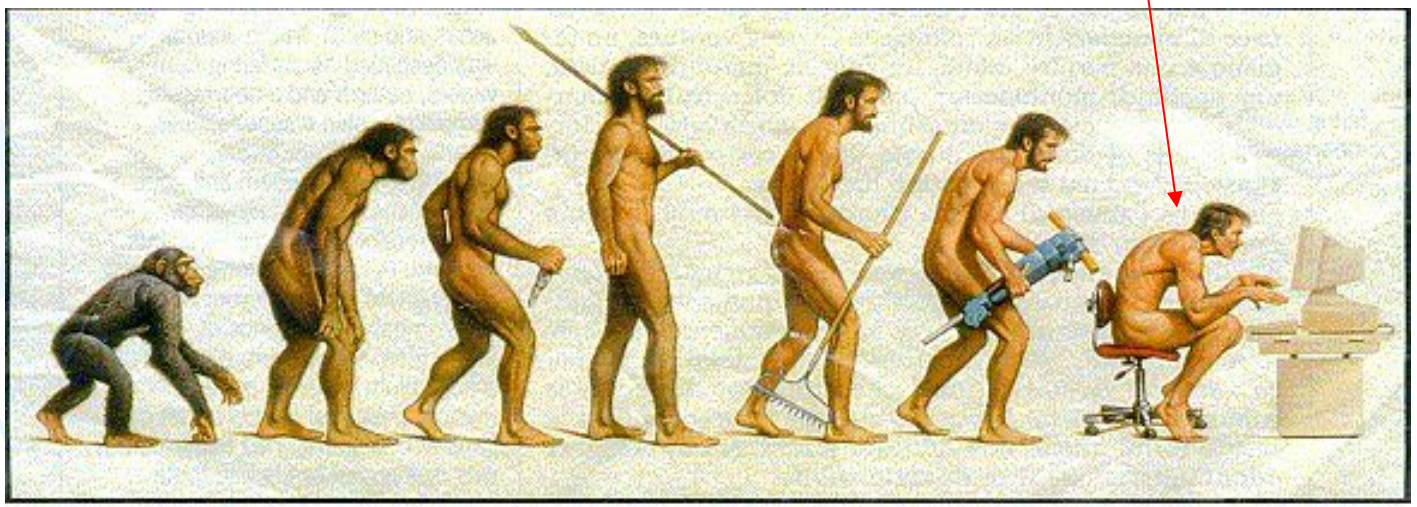
Wildcards

- Some characters (called *wildcards*) mean special things to the shell
- * matches zero or more characters
  - So **ls bio/*.txt** lists all the text files in the bio directory

```
$ ls bio/*.txt
bio/albus.txt bio/ginny.txt bio/harry.txt bio/hermione.txt bio/ron.txt
```

- ? matches any single character
  - So **ls jan-??.txt** lists text files whose names start with "jan-" followed by two characters
  - You can probably guess what ls jan-??.* does
- Note
  - The shell expands wildcards, not individual applications
    - **ls** can't tell whether it was invoked as **ls *.txt** or as **ls earth.txt venus.txt**
  - Wildcards only work with filenames, not with command names
    - ta* does *not* find the tabulate command

Humour



Somewhere, something went terribly wrong

Redirection

- A running program is called a *process*
- Every process automatically has three connections to outside world:
  - *Standard input* (stdin): connected to the keyboard
  - *Standard output* (stdout): connected to the screen
  - *Standard error* (stderr): also connected to the screen
    - Used for error messages
- You can tell the shell to connect standard input and standard output to files instead
- **command < input file** reads from input file instead of from the keyboard
  - Don't need to use this very often, because most Unix commands let you specify the input file (or files) as command-line arguments
- **command > output file** writes to output file instead of to screen
  - Only "normal" output goes to the file, not error messages
- **command < input file > output file** does both

Redirection - examples

- Save number of words in all text files in the tmp directory to words.len:

```
$ cd tmp
$ wc *.txt > words.len
```

- Nothing appears on the screen because output is being sent to the file words.len
- Check contents using cat:

```
$ cat words.len
4   9    69  earth.txt
4   9    69  venus.txt
8  18   138  total
```

- Try typing **cat > junk.txt**
  - No input file specified, so cat reads from the keyboard
  - Output sent to a file

Redirection – things to avoid

- Taking input from the keyboard through **cat** into a file can be viewed as "the world's dumbest text editor"
  - When you're done, use **rm junk.txt** to get rid of the file
- **Don't type rm * unless you're really, really sure that's what you want to do…**
  - Could be the cause of some real heartache!
- Don't redirect out to the same file, *e.g.* **sort words >words**
  - The shell sets up redirection before running the command
  - Redirecting out to an existing file truncates it make it empty
  - sort then goes and reads the empty file
  - Contents of words are lost…

Pipes

- Suppose you want to use the output of one program as the input of another
- e.g., use **wc -w *.txt** to count the words in some files, then **sort -n** to sort numerically
- The obvious solution is to send output of first command to a temporary file, then read from that file

```
$ wc -w *.txt > words.tmp
$ sort -n words.tmp
9 earth.txt
9 venus.txt
$ rm words.tmp
```

- The *right* answer is to use a *pipe*
  - Written as "|"
- Tells the operating system to connect the standard output of the first program to the standard input of the second

```
$ wc -w *.txt | sort -n
9 earth.txt
9 venus.txt
18 total
```

Pipes can give you great flexibility

- More convenient and less error prone than temporary files
- Can chain any number of commands together
  - and combine with input and output redirection

```
$ grep 'Title' spells.txt | sort | uniq -c | sort -n -r | head -10 >
    popular_spells.txt
```

- Any program that reads from standard input and writes to standard output can use redirection and pipes
- Such programs are often called *filters*
- If your programs work like filters, you (and other people) can combine them with standard tools
  - A combinatorial explosion of goodness

Recommended exercises

- Go to the software carpentry home page
  - http://www.swc.scipy.org
- Try the exercises on the “Shell Basics” page
- Indeed I recommend working through all the exercises at some point in your career if you expect to do a lot of programming

10,000 ft view of Programming

- I'll just focus on FORTRAN
  - FORMula TRANslator (1957)
  - Evolved from FORTRAN I, to F66, F77, F90, F95, F2000,… Soon to be F2008
- Old programming style (procedural) that has been modified in later versions to support newer ideas such as Object Oriented Programming
  - Still FORTRAN is little used outside science
- You are free to supply solutions in any language you want but they must work with any data files I supply for questions and run on the departmental Sun machines

Programming Steps

1) Design algorithm

Brainstorm on a board if you like, but write something down in pen and paper

2) Translate this algorithm to FORTRAN

At minimum you should be able to implement the algorithm using FORTRAN commands

Develop the code in an editor

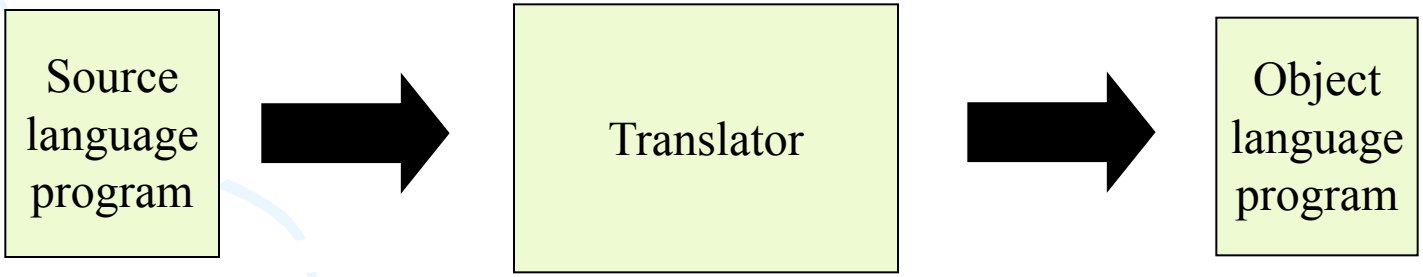
If you can test each subroutine

3) Compile the program

4) Execute

A note on converting languages

- All languages (except machine code) must be translated into machine instructions by some process
  - The only language that is not translated is machine code
  - Any program that converts one language into another is a *translator*

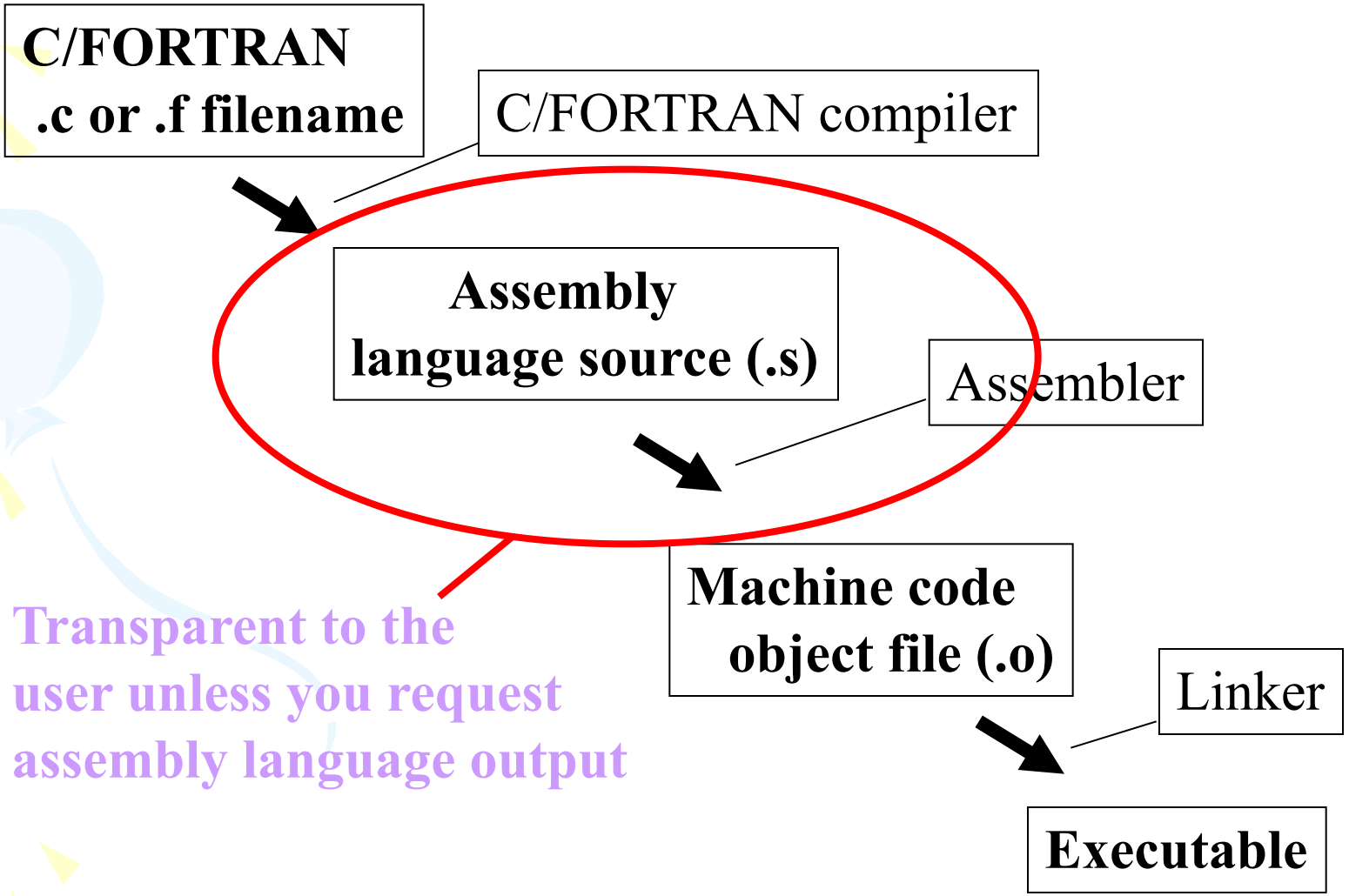


The process of translating to machine code can take several steps through intermediate languages.

Assemblers vs Compilers

- The translator for assembly->machine code is called an *assembler*
- The C *compiler* (for example) is effectively a translator from C to machine code
  - But! Some compilers step through intermediate generation of assembly that you can see
    - The resultant code is then passed to an assembler
    - This step may or may not be visible
- High level language compilers are considerably harder to design than an assembler
  - More abstraction requires the compiler designer work harder

Compilation stages exposed



Desired Properties of Programs

- Efficient
  - Algorithm should be optimally programmed
  - Use memory effectively
- Readability (your programming style)
  - Vertical tabbed alignment
  - Easy mnemonics for variable names (this is quite important!)
  - Well commented
- Generality
  - Flexible inputs (not always possible though)
  - Adaptable – pieces can be used in other codes

Summary

- You should now be able to move around Unix directories
  - List files, change permissions and edit
- You should now understand paths, both relative and absolute
- Know how to use redirection & pipes