Organization of a computer system

- Users
- Applications
- Graphical user interface (GUI)
- Shell
- Operating system
- Hardware (or software acting like hardware: “virtual machine”)
Organization of a computer system

user computer

network

server

“The Internet”

lectura.cs.arizona.edu
Organization of a computer system

Easier to use; Not so easy to program with, automate

system calls

graphical user interface (GUI)

interactive actions (click, drag, tap, …)

users

applications

operating system

hardware
(or software acting like hardware: “virtual machine”)

shell
Easier to program with and automate; Not so convenient to use (maybe)

Users

Applications

Typed commands

Graphical user interface (GUI)

System calls

Operating system

Hardware

(or software acting like hardware: “virtual machine”)
Organization of a computer system

- Users
- Applications
- Graphical user interface (GUI)
- Shell
- Operating system (e.g., Windows, X, Linux)
- Hardware (or software acting like hardware: "virtual machine")
Reading

- Chapter 1:
  - Upto “Background Jobs” (page 17)
What is Unix?

• Unix is an operating system
  – sits between the hardware and the user/applications
  – provides high-level abstractions (e.g., files) and services (e.g., multiprogramming)

• Linux:
  – a “Unix-like” operating system: user-level interface very similar to Unix
  – code base is different from original Unix code
Layers of a Unix system

- Users
- Applications
- Shell
- Unix operating system kernel
- Hardware

System calls
### Unix Commands

- Each command performs [variations of] a single task
  - “options” can be used to modify what a command does
  - different commands can be “glued together” to perform more complex tasks

### Syntax:

```
    command     options     arguments
```

**Examples:**

<table>
<thead>
<tr>
<th>Command</th>
<th>Options</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cd</td>
<td></td>
<td>/home/debray</td>
</tr>
<tr>
<td>ls</td>
<td>-a -l</td>
<td>/usr/local</td>
</tr>
</tbody>
</table>

Options can (usually) be combined together: these are equivalent.
Unix Commands

• Each command performs [variations of] a single task
  – “options” can be used to modify what a command does
  – different commands can be “glued together” to perform more complex tasks

• Syntax:

  \[
  \text{command} \quad \text{options} \quad \text{arguments}
  \]

Examples:

<table>
<thead>
<tr>
<th>Command</th>
<th>Options</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cd</td>
<td></td>
<td>/home/debray</td>
</tr>
<tr>
<td>ls</td>
<td>-a -l</td>
<td>/usr/local</td>
</tr>
<tr>
<td>ls</td>
<td>-al</td>
<td></td>
</tr>
</tbody>
</table>

Not always required: may have default values

typical defaults:
• input: stdin
• output: stdout
• directory: current

defaults to current directory
Examples of Unix commands I

• Figuring out one’s current directory:  **pwd**

• Moving to another directory:  **cd**  *targetdir*

  *Examples:*

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd /</td>
<td>move to the root of the file system</td>
</tr>
<tr>
<td>cd ~</td>
<td>move to one’s home directory</td>
</tr>
<tr>
<td></td>
<td>(also: just “cd” by itself)</td>
</tr>
<tr>
<td>cd /usr/local/src</td>
<td>move to /usr/local/src</td>
</tr>
<tr>
<td>cd ../..</td>
<td>move up two levels</td>
</tr>
</tbody>
</table>
Examples of Unix commands II

• Command: `ls` — *lists the contents of a directory*
  
  – Examples:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ls</code></td>
<td>list the files in the current directory</td>
</tr>
<tr>
<td></td>
<td><em>won’t show files whose names start with ‘.’</em></td>
</tr>
<tr>
<td><code>ls /usr/bin</code></td>
<td>list the files in the directory <code>/usr/bin</code></td>
</tr>
<tr>
<td><code>ls -l</code></td>
<td>give a “long format” listing (provides additional info about files)</td>
</tr>
<tr>
<td><code>ls -a</code></td>
<td>list all files in the current directory, including those that start with ‘.’</td>
</tr>
<tr>
<td><code>ls -al /usr/local</code></td>
<td>give a “long format” listing of all the files (incl. those starting with ‘.’) in <code>/usr/local</code></td>
</tr>
</tbody>
</table>
Executing commands

Typing a command name at the **bash** prompt and pressing the **ENTER** key causes the command to be executed.

The command's output, if any, is displayed on the screen. Examples:

```
% hostname
lectura.cs.arizona.edu
% whoami
eanson
% true
% date
Sat Aug 15 18:54:39 MST 2015
% ps
   PID TTY          TIME CMD
22758 pts/18   00:00:00 bash
30245 pts/18   00:00:00 ps
```
Most commands accept one or more arguments:

```
% cal 9 2015
September 2015
Su Mo Tu We Th Fr Sa
  1  2  3  4  5
  6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

% echo Hello, world!
Hello, world!

% factor 223092870
223092870: 2 3 5 7 11 13 17 19 23
```
Many commands accept *options* that adjust the behavior of the command.

Options almost always begin with a '-' (minus sign). Options are usually specified immediately following the command. For most programs the ordering of options is not significant but that is a convention, not a rule.

**Examples:**

```
% date
Thu Jan 13 02:19:20 MST 2005

% date -u
Thu Jan 13 09:19:22 UTC 2005

% wc Hello.java
      5      14     127 Hello.java

% wc -l -w Hello.java
      5      14 Hello.java
```

We can say that *wc -l -w Hello.java* has two options and one operand.
Whitespace is often significant in command lines. For example, the following commands are all invalid: (Try them!)

```bash
% date-u

% wc -l-w Hello.java

% wc -- notes Hello.java
```
The file system

• A file is basically a sequence of bytes

• Collections of files are grouped into directories (≈ folders)

• A directory is itself a file
  ➔ file system has a hierarchical structure (i.e., like a tree)
    o the root is referred to as “/”
“Everything is a file”

• In Unix, everything looks like a file:
  – documents stored on disk
  – directories
  – inter-process communication
  – network connections
  – devices (printers, graphics cards, interactive terminals, …)

• They are accessed in a uniform way:
  – consistent API (e.g., read, write, open, close, …)
  – consistent naming scheme (e.g., /home/debray, /dev/cdrom)
Referring to files: Absolute Paths

- An **absolute path** specifies how to get to a file starting at the file system root
  - list the directories on the path from the root (“/”), separated by “/”
Referring to files: Absolute Paths

• An **absolute path** specifies how to get to a file starting at the file system root
  – list the directories on the path from the root ("/") separated by "/"

absolute path: `/dd/ee/gg`
Referring to Files: Relative Paths

- Typically we have a notion of a "current directory"
- A relative path specifies how to get to a file starting from the current directory
  - `..` means "move up one level"
  - `.` means current directory
  - list the directories on the path separated by "/"
Referring to files: Relative Paths

• Typically we have a notion of a “current directory”

• A relative path specifies how to get to a file starting from the current directory
  – ‘..’ means “move up one level”
  – ‘.’ means current directory
  – list the directories on the path separated by “/”

Example:
ff relative to ee is: ../ff
Referring to files: Relative Paths

- Typically we have a notion of a "current directory"
- A relative path specifies how to get to a file starting from the current directory
  - '..' means "move up one level"
  - '.' means current directory
  - list the directories on the path separated by "/"

Example:
cc relative to ee is: ../../cc
Home directories

• Each user has a “home directory”
  – specified when the account is created
  – given in the file /etc/passwd

• When you log in, your current directory is your home directory

• Notational shorthand:
  – one’s own home directory:  ~
  – some other user joe’s home directory: ~joe
Some commands for dealing with files

• **pwd**
  • *print the name of the current/working directory*

• **ls [file]**
  • list a directory contents

• **cd [dir]**
  • change the current/working directory

• **cp file\_1 file\_2**
  • copy *file\_1* to *file\_2*

• **vi [file]**
  • the vi editor

• **vimtutor**
  • a tutorial for using vi
Input and output

• Data are read from and written to i/o streams
• There are three predefined streams:
  - stdin : “standard input” – usually, keyboard input
  - stdout : “standard output” – usually, the screen
  - stderr : “standard error” – for error messages (usually, the screen)

Other streams can be created using system calls (e.g., to read or write a specific file)
I/O Redirection

• Default input/output behavior for commands:
  – stdin: keyboard; stdout: screen; stderr: screen
• We can change this using I/O redirection:

  \[\text{cmd} < \text{file}\] redirect \text{cmd}’s stdin to read from \text{file}

  \[\text{cmd} > \text{file}\] redirect \text{cmd}’s stdout to \text{file}

  \[\text{cmd} >> \text{file}\] append \text{cmd}’s stdout to \text{file}

  \[\text{cmd} &> \text{file}\] redirect \text{cmd}’s stdout and stderr to \text{file}

  \[\text{cmd}_1 | \text{cmd}_2\] redirect \text{cmd}_1’s stdout to \text{cmd}_2’s stdin
Combining commands

• The output of one command can be fed to another command as input.
  – Syntax: \textit{command}_1 \mid \textit{command}_2

Example:

\texttt{ls} lists the files in a directory
\texttt{more} \textit{foo} shows the file \textit{foo} one screenful at a time
\texttt{ls} \mid \texttt{more} lists the files in a directory one screenful at a time

\textbf{How this works:}
• \texttt{ls} writes its output to its \texttt{stdout}
• \texttt{more}'s input stream defaults to its \texttt{stdin}
• the pipe connects \texttt{ls}'s stdout to \texttt{more}'s stdin
• the piped commands run “in parallel”
Finding out about commands I

Figuring out which command to use

- `apropos  keyword`
- `man  –k  keyword`

“searches a set of database files containing short descriptions of system commands for keywords”

- Helpful, but not a panacea:
  - depends on appropriate choice of keywords
    - may require trial and error
  - may return a lot of results to sift through
    - pipe through `more`
Finding out about commands II

Figuring out how to use a command

`man command`

“displays the on-line manual pages”

• Provides information about command options, arguments, return values, bugs, etc.
Example: “man ls”

```
ls - list directory contents

SYNOPSIS
ls [OPTION]... [FILE]...

DESCRIPTION
List information about the files (the current directory by default). Sort entries alphabetically if none of
-cfnusux nor -sort.
Mandatory arguments to long options are mandatory for short options too.

-a, --all
do not ignore entries starting with .
-A, --almost-all
do not list implied . and ..

--author
with -l, print the author of each file

-b, --escape
print octal escapes for nongraphic characters

--block-size=SIZE
use SIZE-byte blocks

-B, --ignore-backups
do not list implied entries ending with -

-c with -lt: sort by, and show, ctime (time of last modification of file status information) with -l: show
ctime and sort by name otherwise: sort by ctime

-C list entries by columns

--color[WHEN]
control whether color is used to distinguish file types. WHEN may be 'never', 'always', or 'auto'
```
Example: “man man”
Example: “man man”

we can specify what kind of information we want
Some other useful commands

- **wc** [file]
  - *word count*: counts characters, words, and lines in the input

- **grep** pattern [file]
  - select lines in the input that match *pattern*

- **head** –n [file]
  - show the first *n* lines of the input

- **tail** –n [file]
  - show the last *n* lines of the input

- **cp** file₁ file₂
  - copy file₁ to file₂

- **mv** file₁ file₂
  - move file₁ to file₂
The diff command

- The diff command looks for differences in files
- You will probably want to know this command since it will be used in grading.
- Down the line you will be doing programming homework.
- An correct executable will be given.
- You'll want to use I/O redirection and the diff command to make sure the output of your code matches the output of the given program.
Odds and ends

• Here are some handy options for `ls`:
  • `–t` Sort by modification time instead of alphabetically.
  • `–h` Show sizes with human-readable units like K, M, and G.
  • `–r` Reverse the order of the sort.
  • `–S` Sort by file size
  • `–d` By default, when an argument is a directory, `ls` operates on the entries contained in that directory. `–d` says to operate on the directory itself. Try "`ls -l .`" and "`ls -ld .`".
  • `-R` Recursively list all the subdirectories.
• There are many more and you might want to look at the man page and play
Odds and ends

- Two handy options for `cp`:
  - `R` Recursively copy an entire directory tree
  - `p` Preserve file permissions, ownerships, and timestamps
Odds and Ends

• Many non-alphanumeric characters have special meaning to shells.

• Characters that have special meaning are often called *metacharacters*. Here are the **bash** metacharacters:

```
~ ` ! # $ & * ( ) \ | { } [] ; ' " < > ?
```
Odds and Ends

• Many non-alphanumeric characters have special meaning to shells.

• Characters that have special meaning are often called *metacharacters*. Here are the *bash* metacharacters:

• `~` `\` `!` `#` `$` `&` `*` `(` `)` `|` `{` `}` `[]` `;` `'` `"` `<` `>` `?`

• If an argument has metacharacters or whitespace we suppress their special meaning by enclosing the argument in quotes.
Many non-alphanumeric characters have special meaning to shells.

Characters that have special meaning are often called *metacharacters*. Here are the *bash* metacharacters:

~ ` ! # $ & * ( ) \ [ ] { } ; ' " < > ?

If an argument has metacharacters or whitespace we suppress their special meaning by enclosing the argument in quotes.

An alternative to wrapping with quotes is to use a backslash to "escape" each metacharacter.
Pattern matching in the shell

• Some metacharacters are used as patterns in shell commands, e.g.:

- * matches any string
- [ ... ] matches any one of the characters within braces

Example:

- `ls b*c` list files that begin with b and end with c
- `ls a[xyz]*` list files starting with a followed by x, y, or z
- `ls *.pdf` list files ending with “.pdf”
bash supports simple command line recall and editing with the "arrow keys" but many control-key and escape sequences have meaning too. Here are a few:

- ^A/ ^E  Go to start/end of line.
- ^W  Erase the last "word".
- ^U  Erase whole line. (^C works, too.)
- ^R  Do incremental search through previous commands.
- ESC-f/b  Go forwards/backwards a word. (Two keystrokes: ESC, then f)
- ESC- .  Insert last word on from last command line. (Very handy!)

bash also does command and filename completion with TAB:

Hit TAB to complete to longest unique string.
If a "beep", hit TAB a second time to see alternatives.
Getting more information about files

- `ls -l`: provides additional info about files

```
% ls -l
total 228
drwxrwxr-x 11 patrick 29427 4096 2009-09-24 22:27 acm
drwxrwxr-x 31 137 officeweb 4096 2009-12-23 09:50 admin
drwxrws--- 22 gmt dept 4096 2006-10-17 10:03 archives
drwxrwxr-x 3 gmt officeweb 4096 2006-02-06 09:38 _baks
drwxrwxr-x 19 gmt wheel 4096 2009-06-20 03:33 camera
drwxrwxr-x 76 root officeweb 4096 2010-01-06 08:19 classes
drwxrwxr-x 16 gmt officeweb 4096 2009-08-05 16:32 colloquia
drwxrwxr-x 19 gmt wheel 16384 2009-08-24 08:01 computing
drwxrwxr-x 19 gmt officeweb 4096 2009-10-30 14:16 courses
drwxrwxr-x 2 root wheel 4096 2008-09-29 17:38 data
-drwxr-xr-x 1 gmt wheel 0 2007-08-30 13:01 favicon.ico
drwxrwxr-x 4 gmt wheel 4096 2009-04-08 07:41 general
drwxrwxr-x 8 gmt officeweb 4096 2009-12-09 16:38 graduate
drwxrwxr-x 7 gmt wheel 4096 2007-11-18 05:25 groups
drwxrwxr-x 23 gmt icon 4096 2009-11-24 13:17 icon
-drwxr-xr-x 1 jharriso jharriso 3599 2009-12-22 16:41 index.html
drwxrwxr-x 5 gmt officeweb 4096 2008-08-05 11:20 intranet
drwx------ 2 root root 4096 1996-06-07 09:32 lost+found
-drwxr-xr-x 1 ljacobo ljacobo 2515 2009-09-18 16:12 Microsoft Office Word 2007.Ink
drwxrwxr-x 4 luiten wheel 4096 2008-01-04 13:39 __mm
-drwxrwxr-x 5 luiten dept 4096 2005-10-04 15:45 MDMIP
-drwxrwxr-x 6 gmt rpm 4096 2007-12-12 15:29 mpd
drwxrwxr-x 2 storkerr root 4096 2007-01-10 08:06 msdnaa
-drwxrwxr-x 10 gmt officeweb 4096 2009-12-16 16:04 news
-drwxrwxr-x 2 gmt officeweb 4096 2008-02-08 09:38 _notes
-drwxrwxr-x 5 gmt officeweb 4096 2009-01-07 14:00 partners
-drwxrwxr-x 1 root root 15 2008-09-30 10:32 patterns -> /cs/wwwpatterns
-drwxrwxr-x 428 root root 20480 2010-01-08 02:14 people
-drwxrwxr-x 6 gmt officeweb 4096 2010-01-12 08:30 personnel
-drwxrwxr-x 4 gmt wheel 4096 2009-08-17 14:21 policies
```
Getting more information about files...

<table>
<thead>
<tr>
<th>owner</th>
<th>group</th>
<th>size</th>
<th>last-modified time</th>
<th>file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>gmt</td>
<td>icon</td>
<td>4096</td>
<td>2009-11-24 13:17</td>
<td>icon</td>
</tr>
<tr>
<td>jharriso</td>
<td>jharriso</td>
<td>3599</td>
<td>2009-12-22 16:41</td>
<td>index.html</td>
</tr>
<tr>
<td>gmt</td>
<td>officweb</td>
<td>4096</td>
<td>2008-08-05 11:20</td>
<td>intranet</td>
</tr>
<tr>
<td>root</td>
<td>root</td>
<td>4096</td>
<td>1996-06-07 09:32</td>
<td>lost+found</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>no. of hard links</th>
<th>access permissions</th>
<th>file type</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>drwxrwsr-x</td>
<td>normal file</td>
</tr>
<tr>
<td>1</td>
<td>rw-r--r--</td>
<td>directory</td>
</tr>
<tr>
<td>5</td>
<td>drwxrwsr-x</td>
<td>symbolic link</td>
</tr>
<tr>
<td>2</td>
<td>drwx-----</td>
<td></td>
</tr>
</tbody>
</table>
File access permissions

<table>
<thead>
<tr>
<th>Access Permissions</th>
<th>Owner</th>
<th>Group</th>
<th>Others</th>
<th>File Size</th>
<th>Date Created</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>drwxrwsr-x</td>
<td>23</td>
<td>gmt</td>
<td>icon</td>
<td>4096</td>
<td>2009-11-24</td>
<td>icon</td>
</tr>
<tr>
<td>-rw-r--r-</td>
<td>1</td>
<td>jharris</td>
<td>jharris</td>
<td>3599</td>
<td>2009-12-22</td>
<td>index.html</td>
</tr>
<tr>
<td>drwxrwsr-x</td>
<td>5</td>
<td>gmt</td>
<td>officweb</td>
<td>4096</td>
<td>2008-08-05</td>
<td>intranet</td>
</tr>
<tr>
<td>drwx-------</td>
<td>2</td>
<td>root</td>
<td>root</td>
<td>4096</td>
<td>1996-06-07</td>
<td>lost+found</td>
</tr>
</tbody>
</table>

- **r** read
- **w** write
- **x** execute (executable file)
- **-** enter (directory)
- **no permission**
Changing file access permissions

Command:

\texttt{chmod \textit{who} \textit{what} file\_1 \ file\_2 \ldots \ file\_n} \\
\texttt{\in \{r, w, x\}}

\texttt{\in \{a, u, g, o\}}

Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmod u-w foo</td>
<td>remove write permission for user on file foo</td>
</tr>
<tr>
<td>chmod g+rx bar</td>
<td>give read and execute permission to group for bar</td>
</tr>
<tr>
<td>chmod o-rwx *.doc</td>
<td>remove all access permissions for “other users” (i.e., not owner or group members) for *.doc files</td>
</tr>
<tr>
<td>chmod a+rw p*</td>
<td>give read and write permission to everyone for all files starting with p</td>
</tr>
</tbody>
</table>
Pattern matching: grep

GREP(1)

NAME
grep, egrep, fgrep - print lines matching a pattern

SYNOPSIS
  grep [options] PATTERN [FILE...]
grep [options] [-e PATTERN | -f FILE] [FILE...]

DESCRIPTION
Grep searches the named input FILES (or standard input if no files are named, or the
file name - is given) for lines containing a match to the given PATTERN. By default,
grep prints the matching lines.

In addition, two variant programs egrep and fgrep are available. Egrep is the same as
grep -E. Fgrep is the same as grep -F.

OPTIONS
  -A NUM, --after-context=NUM
    Print NUM lines of trailing context after matching lines. Places a line con-
taining — between contiguous groups of matches.

  -a, --text
    Process a binary file as if it were text; this is equivalent to the --binary-
    files=text option.

  -B NUM, --before-context=NUM
    Print NUM lines of leading context before matching lines. Places a line con-
taining — between contiguous groups of matches.

  -C NUM, --context=NUM
    Print NUM lines of output context. Places a line containing — between contigu-
Pattern matching: grep...

print the current directory

show the contents of this file

print out the lines that match “nation”
Pattern matching: grep...

"print all lines in the input that match the string er"
Pattern matching: grep...

```
% ls | grep -E 'er|re'
```

“print all lines in the input that match the string **er** or **re**

```
% ls | grep -E '^[er|re]'
```

print all lines in the input that begin with the string **er** or **re**
Foreground and Background Processes

• Multiple processes can run concurrently
  – at any point, there is exactly one process that you can interact with through the keyboard (“foreground process”)
  – remaining processes execute “in the background”

• A process can be started in the background:
  processName &

• The execution of the current foreground process can be paused via ctrl-z
  – “bg” will then start it executing in the background
  – “fg” will bring it to the foreground
Example: Deleting a file

Figuring out which command to use:
  – `apropos delete`
    • produces many screenfuls of output that go by too quickly
  – `apropos delete | more`
    • many screenfuls of output, but shown one screenful at a time
    • most of the commands shown aren’t relevant
Example: Deleting a file...

Idea 1: filter out irrelevant stuff

```
man –k delete | grep file
```

(1) a lot fewer results; nothing relevant
Example: Deleting a file...

Idea 2: try a different keyword

`man -k remove | grep file`
Example: Deleting a file...

Idea 2: try a different keyword

`man -k remove | grep file`

these are the only commands that refer to removing files
Example: Deleting a file...

Idea 2: try a different keyword

man –k remove | grep file

this is the only user command that refers to removing files
Example: Deleting a file...

Confirm that this is the appropriate command: "**man rm**"
Setting defaults for your commands

• Create an “alias” for your command
  – syntax different for different shells
  – bash: alias aliasName="cmdName"
    
    e.g.: alias rm="rm –i"
  
  – see “man alias” for details

• To have this alias in force whenever you log in, add this line to the file
  ~/.bashrc   // assuming your login shell is “bash”

• To find out your login shell, run the command
  echo $0
Setting defaults for your commands

• Create an “alias” for your command
  – syntax different for different shells
  – bash: `alias aliasName="cmdName"
    
    e.g.: `alias rm="rm -i"
  
  – see “man alias” for details

• To have this alias in force whenever you log in, add this line to the file
  
  ~/.bashrc  // assuming your login shell is “bash”

  Why didn't this work!!!

• To find out your login shell, run the command

  echo $0
Customization

- If bash is specified as your shell in `/etc/passwd` and you login, the instance of bash that's started is said to be a login shell.

- When `bash` is started as a login shell it first reads `/etc/profile`. It then looks for three files in turn: `~/.bash_profile`, `~/.bash_login`, and `~/.profile`. Upon finding one, it executes the commands in that file and doesn't look any further.

- Sometimes you'll want to start another instance of bash from the bash prompt:
  - `% bash`
  - `%`

- Such an instance of bash is an "interactive non-login shell". It reads `/etc/bash.bashrc` and `~/.bashrc`. 
Customization

• So how do I get around this?
Customization

• So how do I get around this?

1. I could add the line to the file that runs when I log in (in this case .profile)
Customization

• So how do I get around this?
  1. I could add the line to the file that runs when I log in (in this case .profile)
     1. The problem with this is it won't be defined if I invoke another copy of bash
Customization

• So how do I get around this?
  
  1. I could add the line to the file that runs when I log in (in this case .profile)
     
     1. The problem with this is it won't be defined if I invoke another copy of bash
  
  2. I could add the line to both .profile and .bashrc
Customization

• So how do I get around this?

1. I could add the line to the file that runs when I log in (in this case .profile)
   1. The problem with this is it won't be defined if I invoke another copy of bash

2. I could add the line to both .profile and .bashrc
   1. This is OK, but I need to remember to change 2 files every time I change things
Customization

• So how do I get around this?

1. I could add the line to the file that runs when I log in (in this case .profile)
   1. The problem with this is it won't be defined if I invoke another copy of bash

2. I could add the line to both .profile and .bashrc
   1. This is OK, but I need to remember to change 2 files every time I change things

3. I could make a file run in both situations.
   1. This is what most programmers do.
Making `.bashrc` run always

- **Note:** Please use caution when altering hidden files.
- Use `cd` with no arguments to go to your home directory.
- Confirm that you've got `.profile` but not `.bash_profile` or `.bash_login`
- Make a directory `bashoriginals` and copy (cp) `.profile` and `.bashrc` into it.
- Edit `.profile` and add the line:
  
  ```bash
  source ~/.bashrc
  ```
- Now any lines you add to `.bashrc` will run every time a new shell is created.
A key element of the UNIX philosophy is to use **pipelines** to combine programs to solve a problem, rather than writing a new program.

**Problem:** How many unique users are on lectura?

<table>
<thead>
<tr>
<th>Step</th>
<th>Command</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1:</td>
<td>`who</td>
<td>cut -f1 -d &quot; &quot;`</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dmr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ken</td>
</tr>
<tr>
<td></td>
<td></td>
<td>francis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rob</td>
</tr>
<tr>
<td></td>
<td></td>
<td>walt24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dmr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rob</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wnj</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dmr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ken</td>
</tr>
<tr>
<td>v2:</td>
<td>`who</td>
<td>cut -f1 -d &quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sort</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dmr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dmr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>francis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ken</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ken</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ken</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rob</td>
</tr>
<tr>
<td></td>
<td></td>
<td>walt24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wnj</td>
</tr>
<tr>
<td>v3:</td>
<td>`who</td>
<td>cut -f1 -d &quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dmr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>francis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ken</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rob</td>
</tr>
<tr>
<td></td>
<td></td>
<td>walt24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wnj</td>
</tr>
<tr>
<td>v4:</td>
<td>`who</td>
<td>cut -f1 -d &quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sort</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>