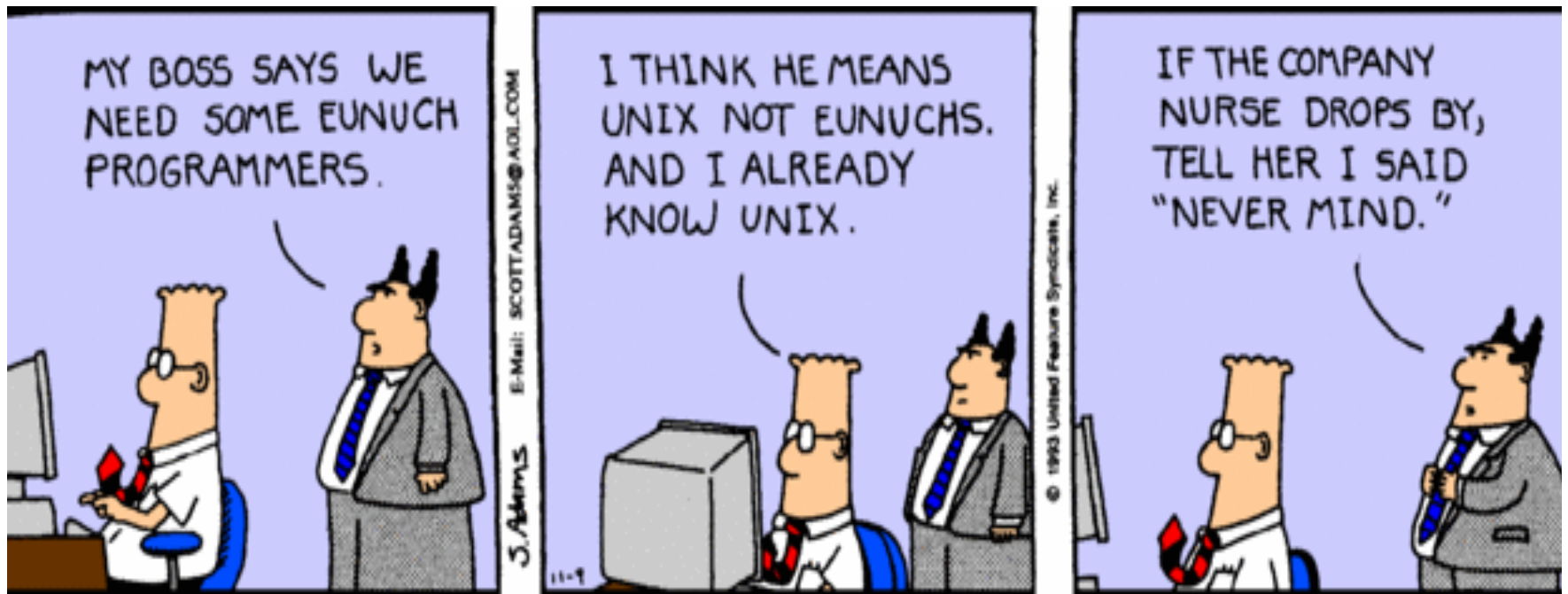


Unix Basics

Katherine Holcomb

UVACSE



Dilbert, November 9, 1993

The Graphical Interface

- Unix uses the X11 graphical system to display the desktop. With NX you are using a different protocol to export the desktop to your computer, but the desktop is displayed with X11. You need a 3-button mouse to use X effectively (the scroll button is nearly always a middle button).
- X11 cut and paste:
 - Using the *left* button, click at the beginning of the text and drag to highlight what you want to cut
 - Move to the other place and click to make sure your cursor is marked with the I-beam
 - Click the *middle* button to paste the text.
- Many recent X11 apps also respond to the Windows control-X/control-C/control-V sequences.

The Shell

- You interact with the operating system via a *shell*. There are several shells available.
- You will have a **prompt** which indicates that the shell is ready to accept commands. It may appear as a \$ or a directory name, or other symbols.
- To find out your shell, at the prompt type
`echo $SHELL`
- Unix in general and the shell in particular is *case-sensitive*.
- The syntax of commands is not completely standardized but in general is
`cmd -o --opts <filename>`

Changing your Shell

- If `echo $SHELL` reveals you are using `ksh` and you'd prefer `bash`, log on to `blue.unix.virginia.edu` and type
`chsh`
(change shell)
- Your NIS password is your normal blue/cluster password.
- Type `/bin/bash` for the new shell. Type carefully; if you mistype you may not be able to log in at all.

Bash History Mechanism

- When using bash you may use its built-in history mechanism to save yourself some keystrokes.
 - Up arrow: scroll through the previous commands you have typed
 - Down arrow: if scrolled back, scroll to more recent commands
 - Left/right arrows: edit text on a line
 - Substitution

`^pattern^newpattern`

substitutes the first occurrence of pattern on the previous command with newpattern. Example:

`./myesec`

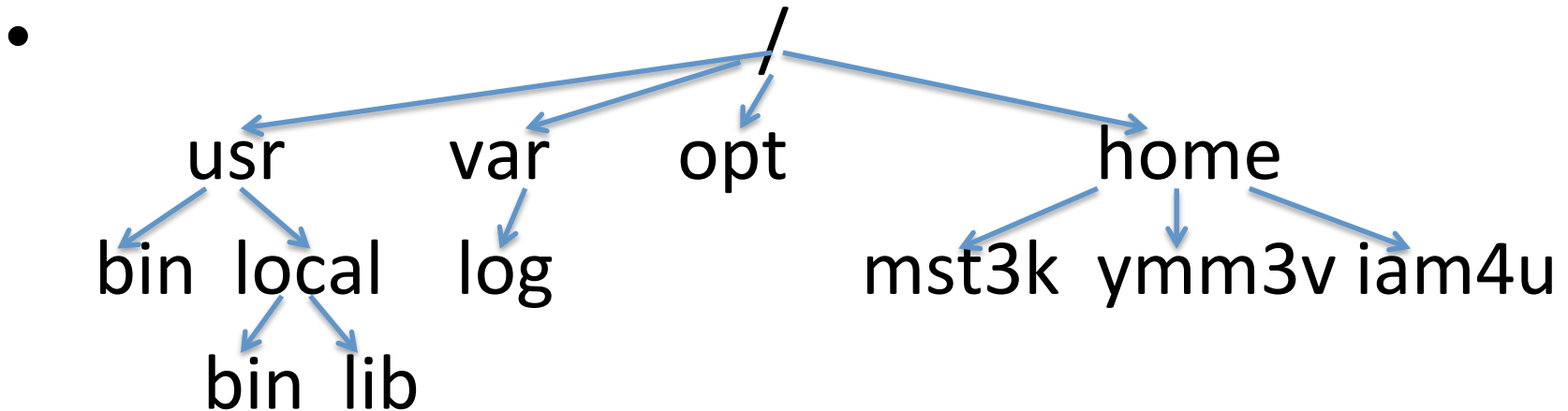
`^esec^exec`

More Bash Goodies

- Tab completion
 - Typing `string<tab>` causes bash to expand `string` further as far as it has a unique name
- Search for earlier command:
`control-r <text>`
- Flip last two characters typed
`control-t`
e.g.
`mro<^t>mor`

The Filesystem

- The Unix filesystem forms a *tree* structure that starts from the *root*, represented by / (forward slash). Files are organized into *directories*.



Paths

- Files are specified by their paths through the directories. Directories are separated by forward slashes.
- Example:
 `/home/mst3k/Documents/writeup.txt`
 This is an **absolute path**.
- The current directory can be represented by a period (.)
 Thus if we are in
 `/home/mst3k/Documents`
 we can type
 `gedit ./writeup.txt`
 The directory above the current directory in the tree (its parent directory) can be represented by ..
 `gedit ../Download/mymod.py`
 These give us **relative paths**.

Files

- In Unix, essentially everything is treated as a file.
- File extensions are meaningless to the Unix system. They may be important to programs such as compilers or interpreters (e.g. Fortran compilers will look for files ending in .f or .f90, R will look for files ending in .R, and so forth) but the operating system itself does not care.

Moving Around

- We need a few commands to move around in the filesystem.
 - Most Unix commands are two to four letter abbreviations
- `cd`
 - Change directory
 - `cd Documents`
 - `cd` with no directory name puts you into your home directory
 - `cd`

Finding Current Directory

- `pwd`
 - Print working directory
- Example
 - `cd`
 - `pwd`
 - `$/home/mst3k`

Adding a Directory

- The `mkdir` command is *make directory*, followed by the name of the directory.
- You can use absolute paths or relative paths for the directory name.

```
mkdir newcode
```

```
mkdir /home/mst3k/newcode/oldsrc
```

Editing Files

- Most IDEs (integrated development environments, e.g. Matlab's desktop and Eclipse for compiled languages and Java) have a built-in editor.
- If you prefer another editor, try
 - `gedit` (graphical user interface)
 - `gvim` (graphical user interface plus commands)
 - `emacs` (graphical user interface plus commands)
- Or for the ambitious or brave, you can use a strictly command-line editor (edit in your shell)
 - `vim` (requires learning some commands, very keyboard-oriented)
 - `vim filename` (it will create it if it doesn't exist)

Viewing Text Files

- `more` is a *pager*; it will show a page at a time. Type the spacebar to advance one page and enter to advance one line.
`more thisfile`
- In Linux `less` is the same as `more` (yes, the name is a joke).
- `cat` shows the entire file (so it may scroll off your screen)
- `head` shows the first 10 lines.
- `head -n filename` shows first n lines
- `tail` shows the last 10 lines
- `tail -n filename` shows last n lines

Copying Files

- `cp`

(copy)

```
cp oldfile newfile
```

```
cp thisfile ../../place/else/thefile
```

```
cp -R thisdir thatdir
```


Listing Files

- The `ls` command lists files

```
ls
```

```
mywriteup.txt
```

- `ls` has many *options*

```
ls -l
```

```
ls -l -h
```

```
ls -F
```

- Most Unix shell commands have a number of command-line options that modify the behavior or output.

Renaming Files

- `mv`

(move)

```
mv oldname newname
```

```
mv thisfile ../otherdir/thefile
```

```
mv thisdir thatdir
```

Deleting Files

- Use the `rm` (remove) command to delete files.

```
rm myfile
```

- You may find that `rm` is *aliased* to

```
rm -i
```

Which will *inquire* for each file. To override this behavior, type

```
rm -f
```

(force) or else

```
/bin/rm thatfile
```

Deleting Directories

- If the directory is empty you can type
`rmdir thatdir`
- If it is not empty and you really want to remove it, type
`rm -rf thatdir`

Advanced Viewing Commands

- `more`
 - In most implementations you can search in the forward direction with `/<pattern>`
- `cat`
 - `cat <filename>` prints to standard output
 - `cat <file1> >> <file2>` appends file1 to file2
- `head <filename>` prints top of file
 - `head -<n> <filename>` prints first n lines
- `tail <filename>` prints end of file
 - `tail -<n> <filename>` prints the last n lines

Your Best Friend

- The basic documentation for a command can be read from the shell with the `man` (manual) command

```
man ls
```

Wildcards

- Strings of characters may be replaced with wildcards. The asterisk (*) can replace zero to unlimited characters (it does not replace a leading period). The question mark (?) replaces exactly one character. Wildcards enable you to work with files without needing to type multiple files with similar names.

```
ls *.py
```

```
rm list?.sh
```

BE CAREFUL when using wildcards with rm! Gone is gone! (We do back up your ITS home directory, but other directories, such as /bigtmp, may not be backed up.)

Finding Text

- You can search for combinations of characters in your files with `grep` (usually said to stand for *get regular expression and print*)
- Example

```
grep -i write *f90
grep "the quick brown fox" *txt
```


Finding Files

- The `find` command is often very useful
`find <path> -<option> filename`
- Most common usage:
`find . -name myfile`
- On Linux the `.` may be omitted (other Unices, including Mac OSX, require it).

Handy Commands

- Miscellaneous handy commands:
- `which <executable>`
 - indicates the path to the executable specified
- `wc <file>`
 - word count
 - `wc -l <file>` is lines only
- `diff <file1> <file2>`
 - Shows differences between files on a per-line basis
- `exit`
 - exit current shell (if login shell, this logs you off!)

More Handy Commands

- `cut <file>`
 - cuts out selected portions of a line (based on fields separated by a delimiter)
 - `cut -d delim -fC1,C2,C3`
 - `cut -d ' ' -f1 /etc/resolve.conf`
- `sort <file>`
 - sorts lines of a text file, based on command-line options
- `uniq <file>`
 - removes duplicate lines (file must be sorted first)

Standard Streams

- Each executable has associated with it three I/O streams: **standard input**, **standard error**, and **standard output**.
- Normally these streams come from or go to your console (i.e. your shell).
- Most Unix commands read from standard input and/or write to standard output.
- They are often represented as **stdin**, **stderr**, and **stdout**.

Stream Redirection

- You can redirect standard input with <
`mycode < params.txt`
- Redirect standard output with >
`ls -l > filelist`
- Append with >>
`cat file1 >> file`
- Redirection of standard error depends on the shell
- Bash:
`make >& make.out`
Redirects both stdout and stderr to `make.out`

Pipes

- One of the most powerful properties of Unix is that you can **pipe** the standard output of one command into the standard input of another.
- The pipe symbol `|` is above the backslash on most US keyboards
- Example

```
nm -A *.o | grep -i mysub
```

- `nm` lists the symbols in an object file or library while `grep` searches for the pattern (`-i` means ignore case). Here I am looking through some object files to find where a particular procedure is defined.

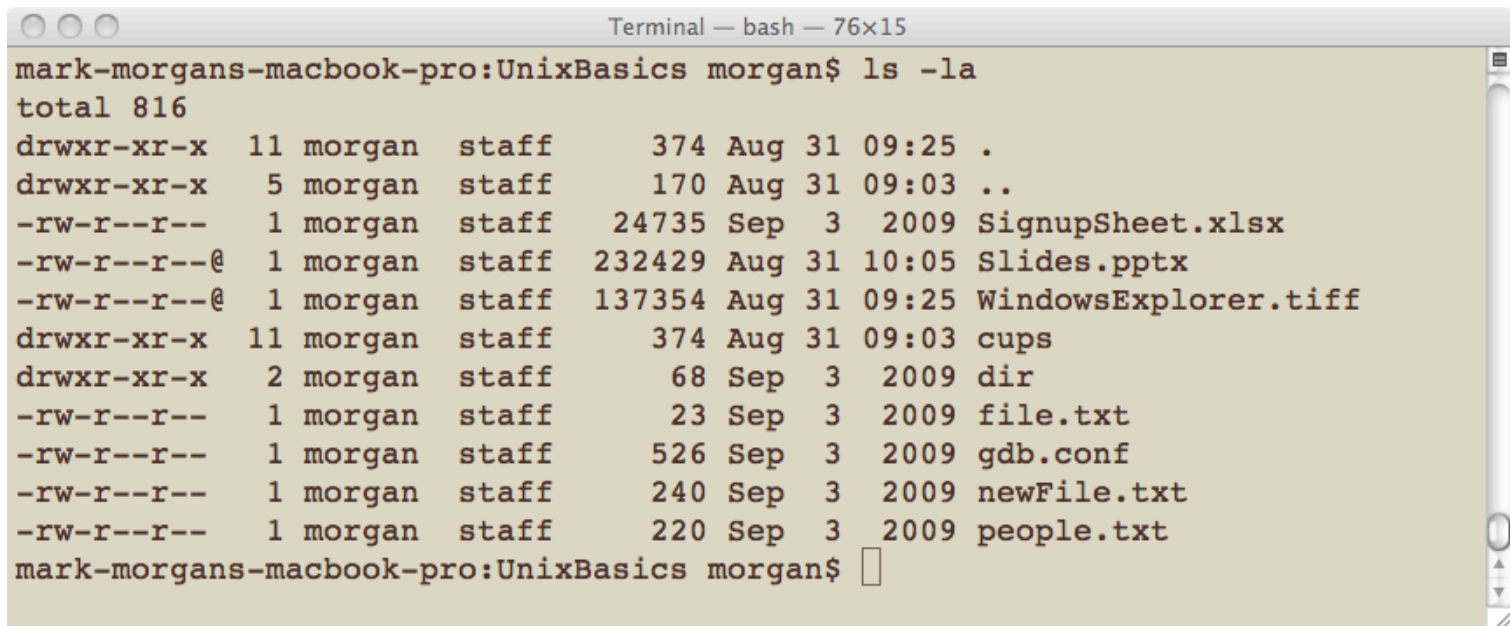
Permissions

- Unix sets permissions on files and directories by *owner, group, and other*.
- Possible permissions are
r, w, x
- File:
r (read), w (write), x (execute)
- Directory:
r (view contents), w (add/remove/modify contents), x (change into directory)

Viewing Permissions

- One of the many options to ls allows us to view permissions:

`ls -l`

A screenshot of a macOS Terminal window. The title bar at the top reads "Terminal — bash — 76x15". The prompt is "mark-morgans-macbook-pro:UnixBasics morgan\$". The command "ls -la" has been entered, and the output is displayed. The output shows a total size of 816 bytes, followed by a list of files and directories with their permissions, owner, group, size, date, and name. The files listed are ".", "..", "SignupSheet.xlsx", "Slides.pptx", "WindowsExplorer.tiff", "cups", "dir", "file.txt", "gdb.conf", "newFile.txt", and "people.txt".

```
mark-morgans-macbook-pro:UnixBasics morgan$ ls -la
total 816
drwxr-xr-x  11 morgan  staff    374 Aug 31 09:25 .
drwxr-xr-x   5 morgan  staff    170 Aug 31 09:03 ..
-rw-r--r--   1 morgan  staff  24735 Sep  3  2009 SignupSheet.xlsx
-rw-r--r--@  1 morgan  staff 232429 Aug 31 10:05 Slides.pptx
-rw-r--r--@  1 morgan  staff 137354 Aug 31 09:25 WindowsExplorer.tiff
drwxr-xr-x  11 morgan  staff    374 Aug 31 09:03 cups
drwxr-xr-x   2 morgan  staff     68 Sep  3  2009 dir
-rw-r--r--   1 morgan  staff     23 Sep  3  2009 file.txt
-rw-r--r--   1 morgan  staff    526 Sep  3  2009 gdb.conf
-rw-r--r--   1 morgan  staff    240 Sep  3  2009 newFile.txt
-rw-r--r--   1 morgan  staff    220 Sep  3  2009 people.txt
mark-morgans-macbook-pro:UnixBasics morgan$
```


ls -l

File Type	User	Group	Other
_	rwX	r_x	r__

Changing Permissions

chmod

- If you are the owner of a file or directory (or you are root/administrator) you can change permissions with chmod
- chmod can use either an octal pattern or a mnemonic pattern. We will mention only the mnemonic:

`chmod [-R] <for whom> ± <permission> <target>`

change [optionally recursive] ugoa ± <rxw> <file /directory>

chmod

- -R changes all permissions through subdirectories
- <who> is user, group, other, or all
- + to add a permission, - to remove it
- permission can be r, w, or x
- target is the name of the file or directory

```
chmod -R a+r mydir
```

```
chmod ug+rw *.c
```

Running Executables

- Executables are often called binaries, especially by Unix types and computer programmers. The terms are synonymous in most cases.
- If the executable is in your *search path* you can simply type its name at the prompt.

```
gedit myfile.py
```

here `gedit` is the name of the binary. Its actual location is `/usr/bin/gedit`, but `/usr/bin` is in the default search path.

- If it is not in your search path you must type the path to the executable (can be absolute or relative)

```
./myprog
```

Usually current directory is not in your default search path for security reasons.

Job Control

- Running from a **command line**:
- Jobs can be running in the *foreground* (no prompt returned to the shell) or *background* (prompt available). To start in the background add an ampersand (&) at the end of the command:

```
./myexec -o myopt myfile &
```

- control-z (ctrl-z or ^z): suspend the job
- bg place into background
- fg forward a backgrounded job

Process Information

- `jobs` shows running jobs
- `ps` shows processes and their IDs (pid)
 - `ps -u <user>` shows processes owned by `user`
 - `ps -f` shows a fuller format
 - `ps -e` shows all processes
 - `ps -ef` all processes, full information

Sample ps Output

```
[kah3f@crestone ~]$ ps -f -u kah3f
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
kah3f	1257	1147	0	Jan09	?	00:00:00	gnome-session
kah3f	1268	1	0	Jan09	?	00:00:00	dbus-launch --sh-syntax --exitwith-se
kah3f	1269	1	0	Jan09	?	00:00:42	/bin/dbus-daemon --fork --print-pid 5

We can cancel processes using the PID or (under Linux) the name of the process

Killing Processes

- control-c (ctrl-c or ^c): kill the current running job (must be foregrounded)
- Find the process ID with ps then:
`kill -9 <pid>` terminates with extreme prejudice
(Linux only) `killall -9 <executable name>` same as above

Environment Variables

- The search path is the set of directories that the operating system will search when you type the name of an executable. To see it, type
`printenv PATH`
- `PATH` is an environment variable. Environment variables describe something about your working environment. Some of them you can set or modify; others are set by the system.
- `printenv`
Prints all environment variables currently set.
- `export VAR`
Allows variable to be passed to child shells

Example

- In most cases, current working directory (.) is not in your default search path. To add it, type (for bash)

```
export PATH=$PATH:.
```

In this case it is essential to add the first `$PATH` or you will lose the default path set by the system.

Sourcing

- Files containing commands can be *sourced*.
This runs the commands in your current shell as if you had typed them at the command line.
- E.g. suppose mymacros consisted of

```
export NVAR=val
export PATH=$PATH:mybins/ACode/the_exec
```

– Then

```
source mymacros
```

sets NVAR and augments PATH

Dotfiles

- “Dotfiles” are files that describe resources to programs that look for them.
- They begin with a period or “dot” (hence the name).
- Normally `ls` does not show them. `ls -a` shows them. Sometimes `ls` is aliased to `ls -a`.
- If you mount a filesystem with dotfiles to Windows and you don’t want to see them, in Windows Explorer go in and set attributes (right-click on file, Properties, Attributes) to Hidden

Bash Dotfiles

- Bash has two
 - `.bash_profile`
 - `.bashrc`
 - if no `.bash_profile` will read `.profile`
- `.bash_profile` is sourced only for login shell
- `.bashrc` is sourced for each file

Aliasing

- You can rename or *alias* executables.. This is especially useful in dotfiles.

```
alias ls='ls -i'
```

```
alias vi='vim'
```

Transferring Files

- You may need to transfer files between your computer and a Unix system. We use the scp (Secure Copy) protocol
- Windows
 - Download SecureFX from Software Central
 - <http://www.itc.virginia.edu/central/>
 - Get SecureCRT also while you're there
- Mac
 - Use the Terminal app in Applications/Utilities

```
scp fir.itc.virginia.edu:myfile ./myfile
```

Some Resources

<http://uvacse.virginia.edu/resources/utilities/unix/unix-intro/>

<http://uvacse.virginia.edu/resources/utilities/unix/unixtut/>

Many, many online tutorials. Google is your friend.