

Shells

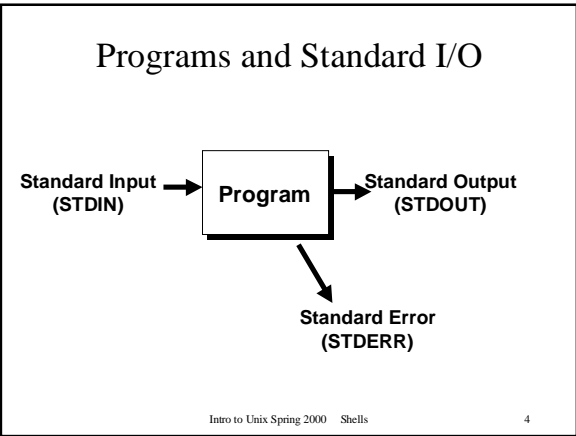
Also known as: Unix Command Interpreter

Shell as a user interface

- A shell is a command interpreter turns text that you type (at the command line) in to actions:
 - runs a program, perhaps the `ls` program.
 - allows you to edit a *command line*.
 - can establish alternative sources of input and destinations for output for programs.

Running a Program

- You type in the name of a program and some command line options:
- The shell reads this line, finds the program and runs it, feeding it the options you specified.
- The shell establishes 3 I/O *channels*:
 - Standard Input
 - Standard Output
 - Standard Error



- ### Unix Commands
- Most Unix commands (programs):
 - read something from standard input.
 - send something to standard output (typically depends on what the input is!).
 - send error messages to standard error.
- Intro to Unix Spring 2000 Shells 5

- ### Defaults for I/O
- When a shell runs a program for you:
 - standard input is your keyboard.
 - standard output is your screen/window.
 - standard error is your screen/window.
- Intro to Unix Spring 2000 Shells 6

Terminating Standard Input

- If standard input is your keyboard, you can type stuff in that goes to a program.
- To end the input you press Ctrl-D (^D) on a line by itself, this ends the input *stream*.
- The shell is a program that reads from standard input.
- What happens when you give the shell ^D?

Intro to Unix Spring 2000 Shells

7

Popular Shells

sh	Bourne Shell
ksh	Korn Shell
cs	C Shell
bash	Bourne-Again Shell

Intro to Unix Spring 2000 Shells

8

Customization

- Each shell supports some customization.
 - User prompt
 - Where to find mail
 - Shortcuts
- The customization takes place in *startup* files – files that are read by the shell when it starts up

Intro to Unix Spring 2000 Shells

9

Startup files

```
sh,ksh:
  /etc/profile (system defaults)
  ~/.profile
bash:
  ~/.bash_profile
  ~/.bashrc
  ~/.bash_logout
csh:
  ~/.cshrc
  ~/.login
  ~/.logout
```

Intro to Unix Spring 2000 Shells

10

Wildcards (metacharacters) for filename abbreviation

- When you type in a command line the shell treats some characters as special.
- These special characters make it easy to specify filenames.
- The shell processes what you give it, using the special characters to replace your command line with one that includes a bunch of file names.

Intro to Unix Spring 2000 Shells

11

The special character *

- * matches anything.
- If you give the shell * by itself (as a command line argument) the shell will remove the * and replace it with all the filenames in the current directory.
- "a*b" matches all files in the current directory that start with **a** and end with **b**.

Intro to Unix Spring 2000 Shells

12

Understanding *

- The **echo** command prints out whatever you give it:

```
> echo hi  
hi
```
- Try this:

```
> echo *
```

Intro to Unix Spring 2000 Shells

13

* and **ls**

- Things to try:

```
ls *  
ls -al *  
ls a*  
ls *b
```

Intro to Unix Spring 2000 Shells

14

Other metacharacters

? Matches any single character

```
ls Test?.doc
```

[abc...] matches any of the enclosed characters

```
ls T[eE][sS][tT].doc
```

[a-z] matches any character in a range

```
ls [a-zA-Z]*
```

[!abc...] matches any character except those listed.

```
ls [!0-9]*
```

Intro to Unix Spring 2000 Shells

15

Input Redirection

- The shell can attach things other than your keyboard to standard input.
 - A file (the contents of the file are fed to a program as if you typed it).
 - A pipe (the output of another program is fed as input as if you typed it).

Output Redirection

- The shell can attach things other than your screen to standard output (or stderr).
 - A file (the output of a program is stored in file).
 - A pipe (the output of a program is fed as input to another program).

How to tell the shell to redirect things

- To tell the shell to store the output of your program in a file, follow the command line for the program with the “>” character followed by the filename:

```
ls > lsout
```

the command above will create a file named **lsout** and put the output of the **ls** command in the file.

Input redirection

- To tell the shell to get standard input from a file, use the “<” character:
`sort < nums`
- The command above would sort the lines in the file `nums` and send the result to `stdout`.

Intro to Unix Spring 2000 Shells

19

You can do both!

```
sort < nums > sortednums  
tr a-z A-Z < letter > rudeletter
```

Intro to Unix Spring 2000 Shells

20

Output and Output Append

- The command `ls > foo` will create a new file named `foo` (deleting any existing file named `foo`).
- If you use `>>` the output will be appended to `foo`:

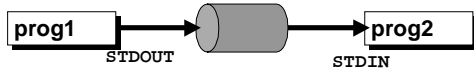
```
ls /etc >> foo  
ls /usr >> foo
```

Intro to Unix Spring 2000 Shells

21

Pipes

- A pipe is a holder for a stream of data.
- A pipe can be used to hold the output of one program and feed it to the input of another.



Intro to Unix Spring 2000 Shells

22

Asking for a pipe

- Separate 2 commands with the “|” character.
- The shell does all the work!

```
ls | sort  
ls | sort > sortedls
```

Intro to Unix Spring 2000 Shells

23

Building commands

- You can string together a series of unix commands to do something new!
- Exercises:
 - List all files in the current directory but only use upper case letters.
 - List only those files that have permissions set so that anyone can write to the file.

Intro to Unix Spring 2000 Shells

24

Stderr

- Many commands send error messages to *standard error*.
 - This is a different *stream* than **stdout**.
- The “>” output redirection only applies to **stdout** (not to **stderr**).
- Try this:

```
ls foo blah gork > savedls
```

(I'm assuming there are no files named foo, blah or gork!).

Intro to Unix Spring 2000 Shells

25

Capturing stderr

- To redirect stderr to a file you need to know what shell you are using.
- When using **sh**, **ksh** or **bash** it's easy:

```
ls foo blah gork 2> erroroutput
```

It's not so easy with csh...

Intro to Unix Spring 2000 Shells

26

File Descriptors

- Unix programs write to *file descriptors*, small integers that are somehow attached to a stream.

```
STDIN is 0
```

```
STDOUT is 1
```

```
STDERR is 2
```

“2>” means redirect stream #2 (**sh**, **ksh** and **bash**)

Intro to Unix Spring 2000 Shells

27

Csh and Stderr

- `>&` merges `STDOUT` and `STDERR` and sends to a file:

```
ls foo blah >& saveboth
```

- `>>&` merges `STDOUT` and `STDERR` and appends to a file:

```
ls foo blah >>& saveboth
```

More Csh

- `|&` merges `STDOUT` and `STDERR` and sends to a pipe:

```
ls foo blah |& sort
```

- To send `STDERR` to file "err" and `STDOUT` to file "out" you can do this:

```
(ls foo blah > out) >& err
```

Shell Variables*

- The shell keeps track of a set of parameter names and values.
- Some of these parameters determine the behavior of the shell.
- We can access these variables:
 - set new values for some to customize the shell.
 - find out the value of some to help accomplish a task.

* From now on I'll focus on sh and ignore csh

Example Shell Variables

sh / ksh / bash

PWD *current working directory*
PATH *list of places to look for commands*
HOME *home directory of user*
MAIL *where your email is stored*
TERM *what kind of terminal you have*
HISTFILE *where your command history
is saved*

Intro to Unix Spring 2000 Shells

31

Displaying Shell Variables

- Prefix the name of a shell variable with "\$".
- The **echo** command will do:

```
echo $HOME  
echo $PATH
```
- You can use these variables on any command line:

```
ls -al $HOME
```

Intro to Unix Spring 2000 Shells

32

Setting Shell Variables

- You can change the value of a shell variable with an assignment command (this is a shell *builtin* command):

```
HOME=/etc  
PATH=/usr/bin:/usr/etc:/sbin  
NEWVAR="blah blah blah"
```

Intro to Unix Spring 2000 Shells

33

set command (shell builtin)

- The **set** command with no parameters will print out a list of all the shell variables.
- You'll probably get a pretty long list...
- Depending on your shell, you might get other stuff as well...

Intro to Unix Spring 2000 Shells

34

\$PS1 and **\$PS2**

- The **PS1** shell variable is your command line prompt.
- The **PS2** shell variable is used as a prompt when the shell needs more input (in the middle of processing a command).
- By changing **PS1** and/or **PS2** you can change the prompt.

Intro to Unix Spring 2000 Shells

35

Fancy **bash** prompts

- Bash supports some fancy stuff in the prompt string:
 - \t is replaced by the current time
 - \w is replaced by the current directory
 - \h is replaced by the hostname
 - \u is replaced by the username
 - \n is replaced by a newline

Intro to Unix Spring 2000 Shells

36

Example **bash** prompt

```
===== [foo.cs.rpi.edu] - 22:43:17 =====  
/cs/hollingd/introunix echo $PS1  
===== [\h] - \t =====\n\w
```

- You can change your prompt by changing PS1:

PS1="Yes Master? "

Intro to Unix Spring 2000 Shells 37

Making Changes Stick

- If you want to tell the shell (**bash**) to *always* use the prompt "**Yes Master ?**", you need to store the change in a *shell startup file*.
- For **bash** - change the file `~/.bashrc`.

- Wait a few minutes and we will talk about text editors - you need to use one to do this!

Intro to Unix Spring 2000 Shells 38

The **PATH**

- Each time you give the shell a command line it does the following:
 - Checks to see if the command is a shell built-in.
 - If not - tries to find a program whose name (the filename) is the same as the command.
- The **PATH** variable tells the shell where to look for programs (non built-in commands).

Intro to Unix Spring 2000 Shells 39

echo \$PATH

```
=====  
[foo.cs.rpi.edu] - 22:43:17 =====  
/cs/hollingd/introunix echo $PATH  
/home/hollingd/bin:/usr/bin:/bin:/usr/local/bin:  
usr/sbin:/usr/bin/X11:/usr/games:/usr/local/  
packages/netscape
```

- The **PATH** is a list of ":" delimited directories.
- The **PATH** is a list and a *search order*.
- You can add stuff to your PATH by changing the shell startup file.

Intro to Unix Spring 2000 Shells

40

Job Control

- The shell allows you to manage *jobs*
 - place *jobs* in the *background*
 - move a job to the foreground
 - suspend a job
 - kill a job

Intro to Unix Spring 2000 Shells

41

Background jobs

- If you follow a command line with "&", the shell will run the *job* in the background.
 - you don't need to wait for the job to complete, you can type in a new command right away.
 - you can have a bunch of jobs running at once.
 - you can do all this with a single terminal (window).

```
ls -lR > saved_ls &
```

Intro to Unix Spring 2000 Shells

42

Listing jobs

- The command `jobs` will list all background jobs:

```
> jobs
[1] Running    ls -lR > saved_ls &
>
```

- The shell assigns a number to each job (this one is job number 1).

Intro to Unix Spring 2000 Shells

43

Suspending and Killing the Foreground Job

- You can suspend the foreground job by pressing `^Z` (Ctrl-Z).
 - Suspend means the job is stopped, but not dead.
 - The job will show up in the `jobs` output.
- You can *kill* the foreground job by pressing `^C` (Ctrl-C).
 - It's gone...

Intro to Unix Spring 2000 Shells

44

Moving a job back to the foreground

- The `fg` command will move a job to the foreground.
 - You give `fg` a job number (as reported by the `jobs` command) preceded by a %.

```
> jobs
[1] Stopped    ls -lR > saved_ls &
> fg %1
ls -lR > saved_ls
```

Intro to Unix Spring 2000 Shells

45

Quoting - the problem

- We've already seen that some characters mean something special when typed on the command line: * ? []
- What if we don't want the shell to treat these as special - we really mean *, not all the files in the current directory:

```
echo here is a star *
```

Intro to Unix Spring 2000 Shells 46

Quoting - the solution

- To turn off special meaning - surround a string with double quotes:

```
echo here is a star "*"
echo "here is a star"
```

Intro to Unix Spring 2000 Shells 47

Careful!

- You have to be a little careful. Double quotes around a string turn the string in to a single command line *parameter*.

```
> ls
fee file? foo
> ls "foo fee file?"
ls: foo fee file?: No such file or directory
```

Intro to Unix Spring 2000 Shells 48

Quoting Exceptions

- Some *special* characters are **not** ignored even if inside double quotes:
- \$ (prefix for variable names)
- " the quote character itself
- \ slash is always something special (\n)
 - you can use \\$ to mean \$ or \" to mean "

```
echo "This is a quote \" "
```

Intro to Unix Spring 2000 Shells

49

Single quotes

- You can use single quotes just like double quotes.
 - Nothing (except ') is treated special.

```
> echo 'This is a quote \" '
This is a quote \"
>
```

Intro to Unix Spring 2000 Shells

50

Backquotes are different!

- If you surround a string with backquotes the string is replaced with the result of running the command in backquotes:

```
> echo `ls`
foo fee file?
> PS1=`date`
Tue Jan 25 00:32:04 EST 2000
```

← new prompt!

Intro to Unix Spring 2000 Shells

51

There is lots more...

- Check the book for more info on job control.
- We also didn't talk about command history, it is very useful:
 - the shell keeps a list of commands that you entered.
 - you tell the shell to repeat the same command again.
