

Crash Course in Unix

For more info check out the Unix man pages

-or-

<http://www.cs.rpi.edu/~hollingd/unix>

-or-

Unix in a Nutshell (an O'Reilly book).

Unix Accounts

- To access a Unix system you need to have an *account*.
- Unix account includes:
 - username and password
 - userid and groupid
 - home directory
 - shell

username

- A username is (typically) a sequence of alphanumeric characters of length no more than 8.
- username is the primary identifying attribute of your account.
- username is (usually) used as an email address
- the name of your home directory is usually related to your username.

password

- a password is a secret string that only the user knows (not even the system knows!)
- When you enter your password the system encrypts it and compares to a stored string.
- passwords are (usually) no more than 8 characters long.
- It's a good idea to include numbers and/or special characters (don't use an english word!)

userid

- a userid is a number (an integer) that identifies a Unix account. Each userid is unique.
- It's easier (and more efficient) for the system to use a number than a string like the username.
- You don't need to know your userid!

Unix Groups and groupid

- Unix includes the notion of a "group" of users.
- A Unix group can share files and active processes.
- Each account is assigned a "primary" group.
- The groupid is a number that corresponds to this primary group.
- A single account can belong to many groups (but has only one primary group).

Home Directory

- A home directory is a place in the file system where files related to an account are stored.
- A *directory* is like a Windows folder (more on this later).
- Many unix commands and applications make use of the account home directory (as a place to look for customization files).

Shell

- A Shell is a unix program that provides an interactive session - a text-based user interface.
- When you log in to a Unix system, the program you initially interact with is your shell.
- There are a number of popular shells that are available.

Logging In

- To log in to a Unix machine you can either:
 - sit at the *console* (the computer itself)
 - access via the net (using telnet, rsh, ssh, kermit, or some other remote access client).
- The system prompts you for your username and password.
- Usernames and passwords are case sensitive!

Session Startup

- Once you log in, your shell will be started and it will display a prompt.
- When the shell is started it looks in your home directory for some customization files.
 - You can change the shell prompt, your PATH, and a bunch of other things by creating customization files.

Your Home Directory

- Every Unix process* has a notion of the “current working directory”.
- Your shell (which is a process) starts with the current working directory set to your home directory.

*A process is an instance of a program that is currently running.

Interacting with the Shell

- The shell prints a prompt and waits for you to type in a command.
- The shell can deal with a couple of types of commands:
 - shell internals - commands that the shell handles directly.
 - External programs - the shell runs a program for you.

Files and File Names

- A file is a basic unit of storage (usually storage on a disk).
- Every file has a name.
- Unix file names can contain any characters (although some make it difficult to access the file).
- Unix file names can be long!
 - how long depends on your specific flavor of Unix

File Contents

- Each file can hold some raw data.
- Unix does not impose any structure on files
 - files can hold any sequence of bytes.
- Many programs *interpret* the contents of a file as having some special structure
 - text file, sequence of integers, database records, etc.

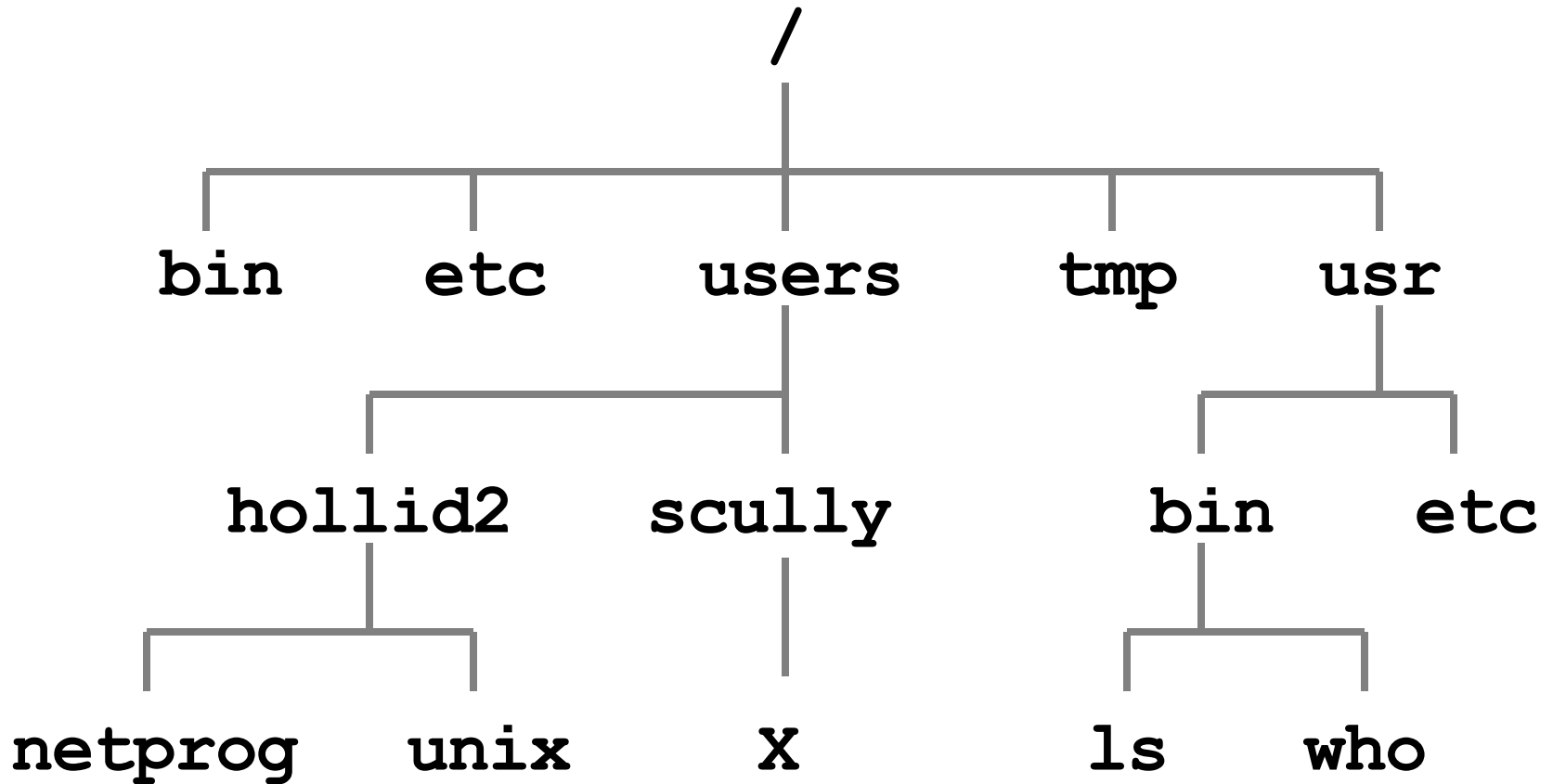
Directories

- A directory is a special kind of file - Unix uses a directory to hold information about other files.
- We often think of a directory as a container that holds other files (or directories).
- Mac and Windows weenies*: A directory is the same idea as a *folder*.
- *weenies is actually a term usually used to describe Unix users - I'm being defensive...

More about File Names

- Review: every file has a name.
- Each file *in* the same directory must have a unique name.
- Files that are in different directories can have the same name.

The Filesystem



Unix Filesystem

- The filesystem is a hierarchical system of organizing files and directories.
- The top level in the hierarchy is called the "root" and *holds* all files and directories.
- The name of the root directory is `/`

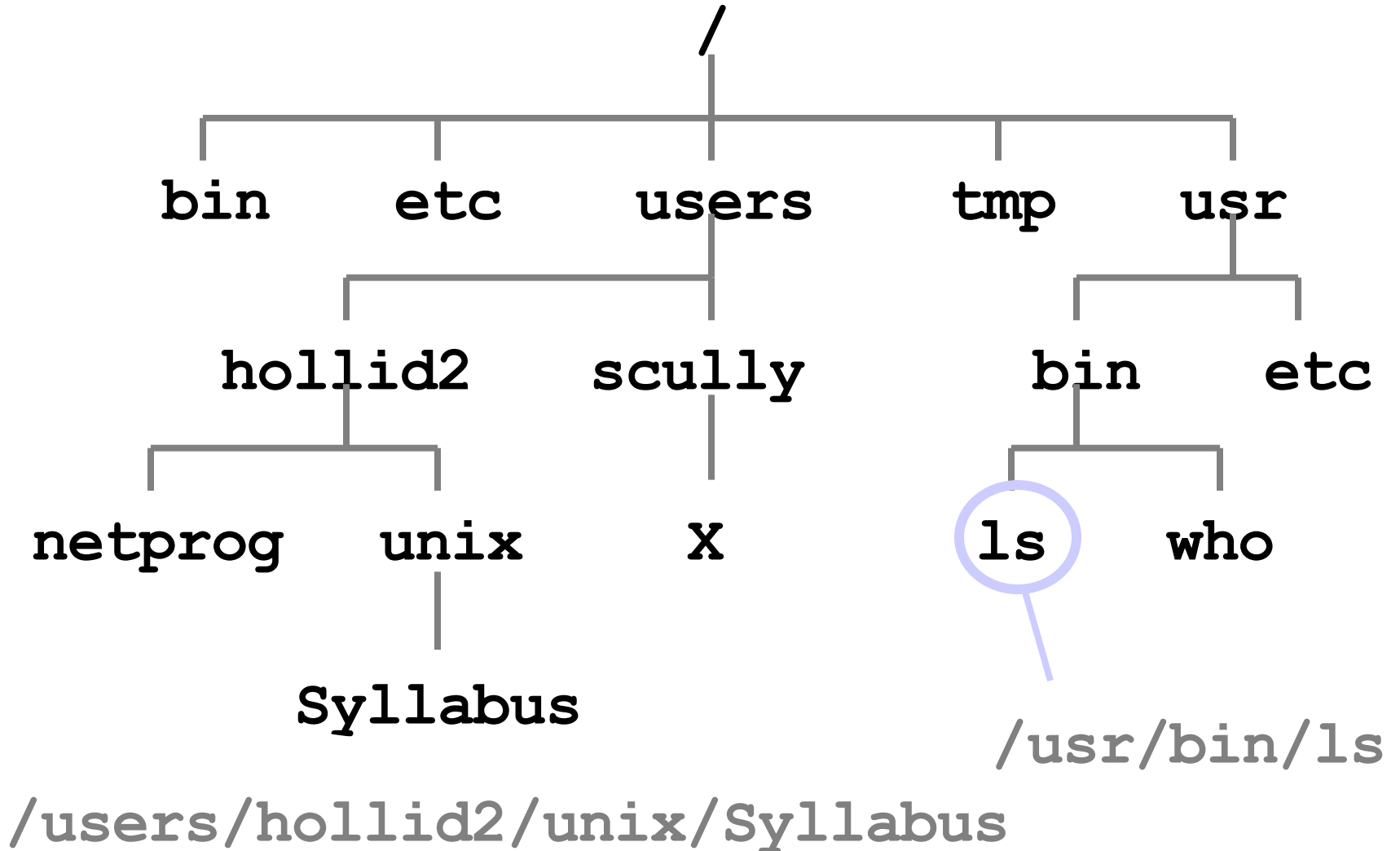
Pathnames

- The *pathname* of a file includes the file name and the name of the directory that holds the file, and the name of the directory that holds the directory that holds the file, and the name of the ... up to the root
- The pathname of every file in a Unix *filesystem* is unique.

Pathnames (cont.)

- To create a pathname you start at the root (so you start with "/"), then follow the path down the hierarchy (including each directory name) and you end with the filename.
- In between every directory name you put a "/".

Pathname Examples



Absolute Pathnames

- The pathnames described in the previous slides start at the *root*.
- These pathnames are called "absolute pathnames".
- We can also talk about the pathname of a file *relative* to a directory.

Relative Pathnames

- If we are *in* the directory `/users/hollid2`, the relative pathname of the file **Syllabus** in the directory `/users2/hollid2/unix/` is:

unix/Syllabus

- Most Unix commands deal with pathnames!
- We will usually use relative pathnames when specifying files.

Example: The ls command

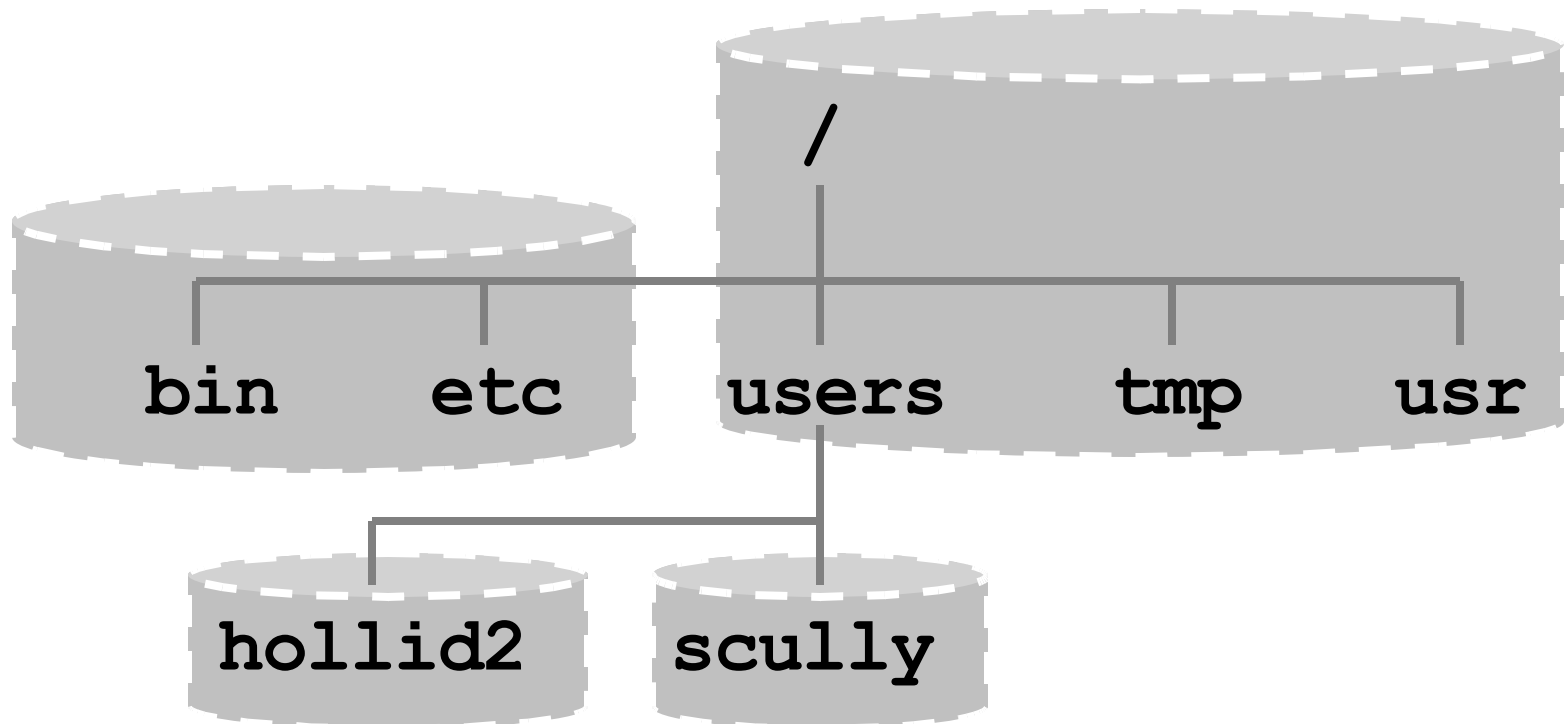
- Exercise: login to a unix account and type the command "ls".
- The names of the files are shown (displayed) as relative pathnames.
- Try this:

```
ls /usr
```

- `ls` should display the name of each file in the directory `/usr`.

Disk vs. Filesystem

- The entire hierarchy can actually include many disk drives.
 - some directories can be on other computers



The current directory and *parent* directory

- There is a special relative pathname for the current directory:

.

- There is a special relative pathname for the parent directory:

..

Some Simple Commands

- Here are some simple commands to get you started:
 - **ls** lists file names (like DOS dir command).
 - **who** lists users currently logged in.
 - **date** shows the current time and date.
 - **pwd** print working directory

The `ls` command

- The **`ls`** command displays the names of some files.
- If you give it the name of a directory as a *command line parameter* it will list all the files in the named directory.

ls Command Line Options

- We can modify the output format of the **ls** program with a *command line option*.
- The **ls** command support a bunch of options:
 - **l** *long* format (include file times, owner and permissions)
 - **a** *all* (shows hidden* files as well as regular files)
 - **F** include special char to indicate file types.

*hidden files have names that start with "."

Moving Around in the Filesystem

- The `cd` command can change the current working directory:

cd *change directory*

- The general form is:

cd [directoryname]

cd

- With no parameter, the **cd** command changes the current directory to your home directory.
- You can also give **cd** a relative or absolute pathname:

```
cd /usr
```

```
cd ..
```

Some more commands and command line options

- **ls -R** will list everything in a directory and in all the subdirectories recursively (the entire hierarchy).
 - you might want to know that Ctrl-C will cancel a command (stop the command)!
- **pwd**: print working directory
- **df**: shows what disk holds a directory.

Copying Files

- The **cp** command copies files:

cp [options] source dest

- The source is the name of the file you want to copy.
- dest is the name of the new file.
- source and dest can be relative or absolute.

Another form of **cp**

- If you specify a dest that is a directory, **cp** will put a copy of the source in the directory.
- The filename will be the same as the filename of the source file.

```
cp [options] source destdir
```

Deleting (removing) Files

- The **rm** command deletes files:

```
rm [options] names...
```

- **rm** stands for "remove".
- You can remove many files at once:

```
rm foo /tmp/blah /users/clinton/intern
```

File attributes

- Every file has some attributes:
 - Access Times:
 - when the file was created
 - when the file was last changed
 - when the file was last read
 - Size
 - Owners (user and group)
 - Permissions

File Time Attributes

- Time Attributes:
 - when the file was last changed `ls -l`
 - when the file was created* `ls -lc`
 - when the file was last read (accessed) `ls -ul`

*actually it's the time the file status last changed.

File Owners

- Each file is owned by a user.
- You can find out the username of the file's owner with the **-l** option to **ls**,
- Each file is also owned by a Unix group.
- **ls -lg** also shows the group that owns the file.

File Permissions

- Each file has a set of permissions that control who can mess with the file.
- There are three kinds of permissions:
 - read abbreviated **r**
 - write abbreviated **w**
 - execute abbreviated **x**
- There are separate permissions for the file owner, group owner and everyone else.

ls -l

```
> ls -l foo
```

```
-rw-rw-----
```

permissions

```
1 hollingd grads 13 Jan 10 23:05 foo
```

owner

group

size

time

name

ls -l and permissions



Type of file:

- means plain file

d means directory

rwx

- Files:

r - allowed to read.

w - allowed to write.

x - allowed to execute

- Directories:

r - allowed to see the names of the files.

w - allowed to add and remove files.

x - allowed to enter the directory

Changing Permissions

- The **chmod** command changes the permissions associated with a file or directory.
- There are a number of forms of chmod, this is the simplest:

chmod mode file

chmod mode file

- Mode has the following form*:

[ugoa] [+ -=] [rwx]

u=user g=group o=other a=all

+ add permission - remove permission = set permission

*The form is really more complicated, but this simple version will do enough for now.

chmod examples

```
> ls -al foo  
rwxrwx--x    1 hollindg grads ...
```

```
> chmod g-wx foo
```

```
> ls -al foo  
-rwxrw----    1 hollindg grads
```

```
> chmod u-r .
```

```
> ls -al foo  
ls: .: Permission denied
```

Other filesystem and file commands

- **mkdir** make directory
- **rmdir** remove directory
- **touch** change file timestamp (can also create a blank file)
- **cat** concatenate files and print out to terminal.

Shells

Also known as: Unix Command Interpreter

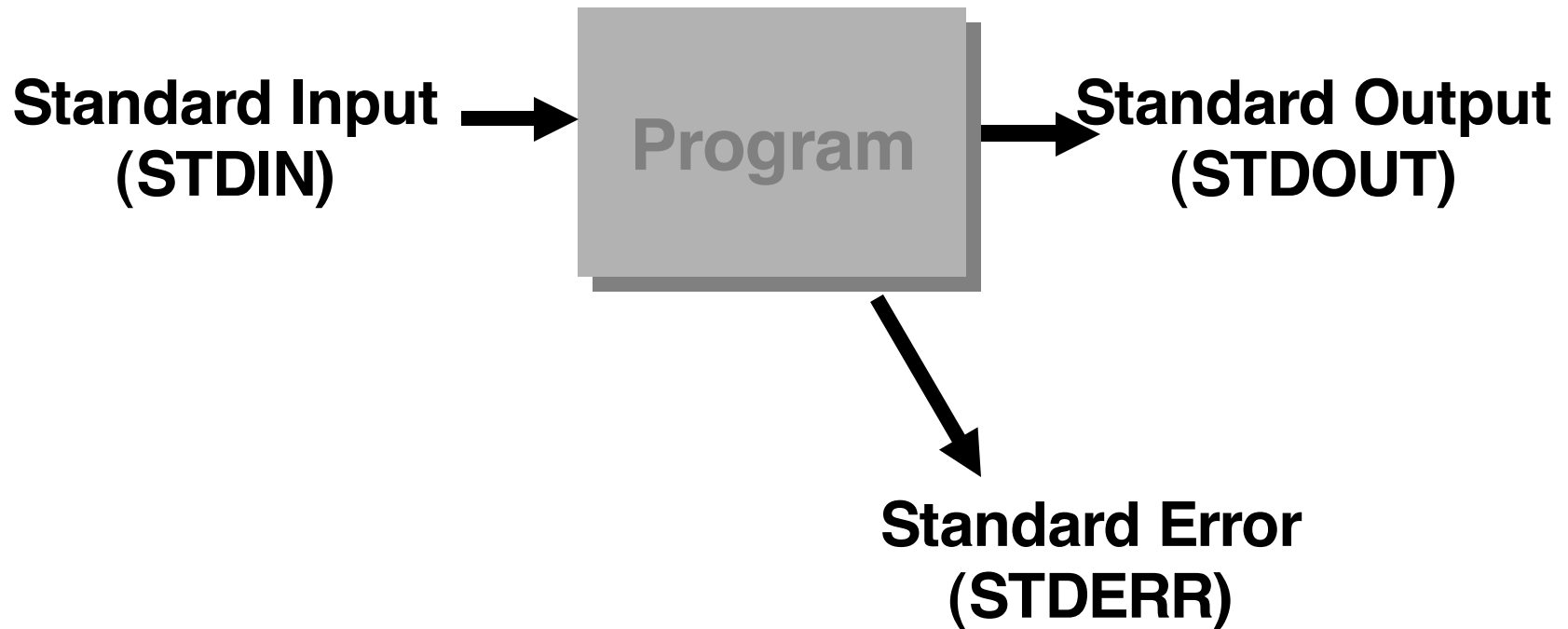
Shell as a user interface

- A shell is a command interpreter that turns text that you type (at the command line) into 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

Programs and Standard I/O



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.

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.

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?

Popular Shells

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

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

Startup files

sh, ksh:

`/etc/profile` (system defaults)

`~/.profile`

bash:

`~/.bash_profile`

`~/.bashrc`

`~/.bash_logout`

csh:

`~/.cshrc`

`~/.login`

`~/.logout`

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.

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**.

Understanding *

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

```
> echo hi
```

```
hi
```

- Try this:

```
> echo *
```

* and **ls**

- Things to try:

```
ls *
```

```
ls -al *
```

```
ls a*
```

```
ls *b
```

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`.

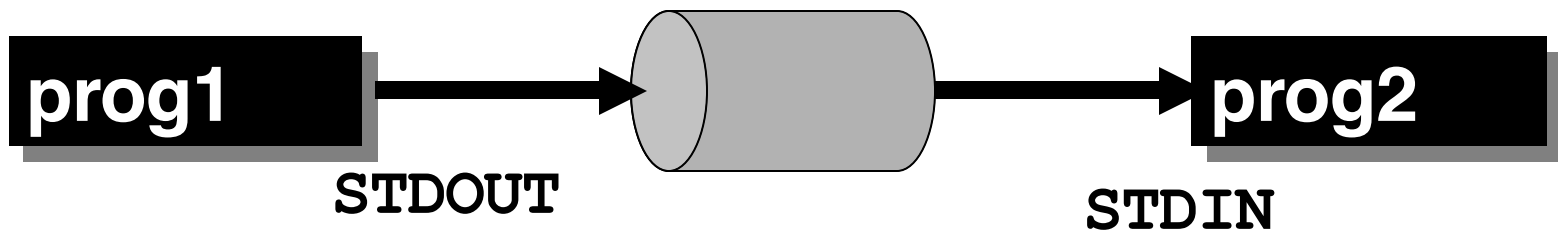
You can do both!

```
sort < nums > sortednums
```

```
tr a-z A-Z < letter > rudeletter
```

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.



Asking for a pipe

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

```
ls | sort
```

```
ls | sort > sortedls
```

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.

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*

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
```

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"
```

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...

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).

echo \$PATH

```
===== [foo.cs.rpi.edu] - 22:43:17 =====  
/cs/hollind/introunix echo $PATH  
/home/hollind/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 (on RCS change ~/ **.bashrc**)

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

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 &
```

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).

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...

Quoting - the problem

- We've already seen that some characters mean something special when typed on the command line: `*` (also `?`, `[]`)
- 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 *
```

Quoting - the solution

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

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

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

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 \" "
```


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!

Programming

- Text editors
 - emacs, vi
 - Can also use any PC editor if you can get at the files from your PC.
- Compilers – gcc is probably best.
- Debuggers: gdb xgdb