

Using Unix

For more info check out the Unix man pages

-or-

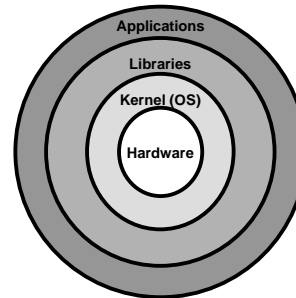
<http://www.cs.rpi.edu/~hollingd/unix>

-or-

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

1

Remember This Picture?



There are many standard applications:

- file system commands
- interactive shells
- text editors
- compilers
- text processing

2

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, kermi, or some other remote access client).
- The system prompts you for your username and password.
- Usernames and passwords are case sensitive!

3

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.

4

Your Home Directory

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

5

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.

6

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

7

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.

8

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 "."

9

Moving Around in the Filesystem

- There **cd** command can change the current working directory:

cd *change directory*

- The general form is:

cd [directoryname]

10

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 ..

11

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.

12

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.

13

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`

14

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`

15

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

16

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.

17

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.

18

Shells

Also known as: Unix Command Interpreter

19

Shell as a user interface

- A shell is a command interpreter that 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.

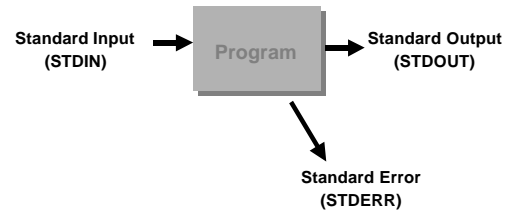
20

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

21

Programs and Standard I/O



22

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.

23

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.

24

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?

25

Popular Shells

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

26

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

27

Startup files

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

28

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.

29

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**.

30

Understanding *

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

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

```
> echo *
```

31

* and **ls**

- Things to try:

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

32

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).

33

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).

34

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.

35

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**.

36

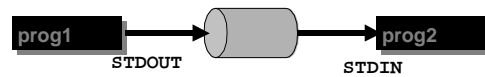
You can do both!

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

37

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.



38

Asking for a pipe

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

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

39

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.

40

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*

41

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
```

42

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"
```

43

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...

44

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).

45

echo **\$PATH**

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

- 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**)

46

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

47

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 &
```

48

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).

49

## 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...

50

## 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 *
```

51

## Quoting - the solution

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

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

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

52

## 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 \" "
```

53

## 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 \"
>
```

54

## 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!

55

## 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

56