

## Using UNIX

This document gives an overview of how to use UNIX.

I don't know how to write this, but it's very useful.

1

## Getting More Help

You can get more information about any of the topics discussed here by typing

"man<topicname>", for instance, → man ls

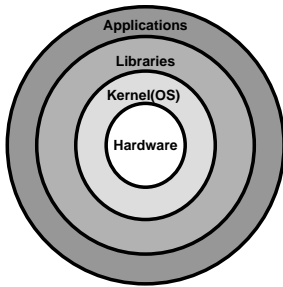
If you don't quite remember the name of the command, you can say

"apropos<topicname>", for instance → apropos directory

There are also many good books in the library that describe all this.

2

## Remember This Picture?



There are many standard applications:

- filesystem commands
- interactive shells
- text editors
- compilers
- text processing

3

## Logging In

- To log into 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!

4

## 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.

5

## 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.

6

## InteractingwiththeShell

- Theshellprintsapromptandwaitsforyou totypeinacommand.
- Theshellcandealwithacoupleoftypesof commands:
  - shellinternals - commandsthattheshell handlesdirectly.
  - Externalprograms - theshellrunsaprogram foryou.

7

## SomeSimpleCommands

- Herearesomesimplecommandstoyou started:
  - **ls** listsfilenames(likeDOSdircommand).
  - **who** listsuserscurrentlyloggedin.
  - **date** showsthecurrenttimeanddate.
  - **pwd** printworkingdirectory

8

## The ls command

- Thelscommanddisplaysthenamesof somefiles.
- Ifyougiveitthennameofadirectoryasa *commandlineparameter* itwilllistallthe filesinthenameddirectory.

9

## ls CommandLineOptions

- Wecanmodifytheoutputformatofthe **ls** program witha *commandlineoption* .
- Thelscommandsupportabunchofoptions:
  - **l** *long* format(includefiletimes,ownerandpermissions)
  - **a** *all* (showshidden\*filesaswellasregularfiles)
  - **F** includespecialchartoindicatefiletypes.

\*hiddenfileshavenamesthatstartwith"."

10

## MovingAroundintheFileSystem

- Therecdcommandcanchangethecurrent workingdirectory:  
**cd** *change directory*
- Thegeneralformis:  
**cd [directoryname]**

11

## cd

- Withnoparameter,the **cd** command changesthecurrentdirectorytoyourhome directory.
- Youcanalsogive **cd** arelativeorabsolute pathname:

```
cd /usr
cd ..
```

12

## Somemorecommandsand commandlineoptions

- **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.

13

## 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.

14

## 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**

15

## 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**

16

## 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

17

## 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.

18

## Other filesystem and file commands

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

19

## Shells

Also known as: Unix Command Interpreter

20

## 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.

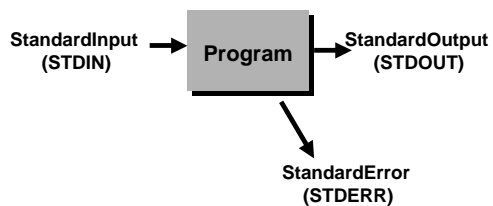
21

## 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

22

## Programs and Standard I/O



23

## Unix Commands

- Most Unix commands (programs):
  - read something from standard input.
  - send something to standard output (typically depend on what the input is!).
  - send error messages to standard error.

24

## DefaultsforI/O

- Whenashellrunsaprogramforyou:
  - standardinputisyourkeyboard.
  - standardoutputisyourscreen/window.
  - standarderrorisyourscreen/window.

25

## TerminatingStandardInput

- Ifstandardinputisyourkeyboard,youcantypestuffinthatgoestoaprogram.
- ToendtheinputyoupressCtrl -D(^D)ona linebyitself,thisendstheinput *stream*.
- Theshellisaprogramthatreadsfrom standardinput.
- Whathappenswhenyougivetheshell^D?

26

## PopularShells

<b>sh</b>	BourneShell
<b>ksh</b>	KornShell
<b>csh</b>	CShell
<b>bash</b>	Bourne-AgainShell

27

## Customization

- Eachshellssupportssomecustomization.
  - Userprompt
  - Wheretofindmail
  - Shortcuts
- Thecustomizationtakesplacein *startup* files – filesthatarereadbytheshellwhenit startsup

28

## Startupfiles

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

29

## Wildcards(metacharacters)for filenameabbreviation

- Whenyoutypeinacommandlinetheshell treatssomecharactersasspecial.
- Thesespecialcharactersmakeiteasyto specifyfilenames.
- Theshellprocesseswhatyougiveit,using thespecialcharacterstoreplaceyour commandlinewithonethatincludesa bunchoffilenames.

30

## 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.

31

## Understanding \*

- The **echo** command prints out whatever you give it:  
    > **echo hi**  
    **hi**
- Try this:  
    > **echo \***

32

## \* and ls

- Things to try:  
    **ls \***  
    **ls -al \***  
    **ls a\***  
    **ls \*b**

33

## 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).

34

## 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 a file).
  - A pipe (the output of a program is fed as input to another program).

35

## 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.

36

## Input redirection

- 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`.

37

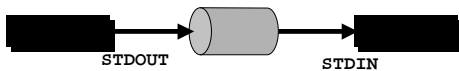
## You can do both!

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

38

## 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.



39

## Asking for a pipe

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

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

40

## 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.

41

## Example Shell Variables

**sh / ksh / bash**

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

42

## 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
```

43

## 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"
```

44

## 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...

45

## The PATH

- Each time you give the shell a command line it does the following:
  - Check to see if the command is a shell built-in.
  - If not - try 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 (nonbuilt-in commands).

46

## echo \$PATH

- 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`)

47

## 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

48

## 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 &
```

49

## 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).

50

## 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...

51

## 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 *
```

52

## Quoting - the solution

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

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

```
> here is a star *
```

53

## 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 \" "
```

54

## 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 \"
>
```

55

## 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!*

56

## Programming

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

57