Introduction to Unix

Rob Funk

<funk+@osu.edu>

University Technology Services

Workstation Support

http://wks.uts.ohio-state.edu/

Course Objectives

- basic background in Unix structure
- knowledge of getting started
- directory navigation and control
- file maintenance and display commands
- shells
- Unix features
- text processing

Course Objectives

Useful commands

- working with files
- system resources
- printing
- vi editor

In the *Introduction to UNIX* document

3

- shell programming
- Unix command summary tables
- short Unix bibliography (also see web site)

We will not, however, be covering these topics in the lecture.

Numbers on slides indicate page number in book.

History of Unix

7–8

1960s	multics project (MIT, GE, AT&T)
1970s	AT&T Bell Labs
1970s/80s	UC Berkeley
1980s	DOS imitated many Unix ideas
	Commercial Unix fragmentation
	GNU Project
1990s	Linux
now	Unix is widespread and available from many sources, both free and commercial

Unix Systems

7–8

SunOS/Solaris	Sun Microsystems
Digital Unix (Tru64)	Digital/Compaq
HP-UX	Hewlett Packard
Irix	SGI
UNICOS	Cray
NetBSD, FreeBSD	UC Berkeley / the Net
Linux	Linus Torvalds / the Net

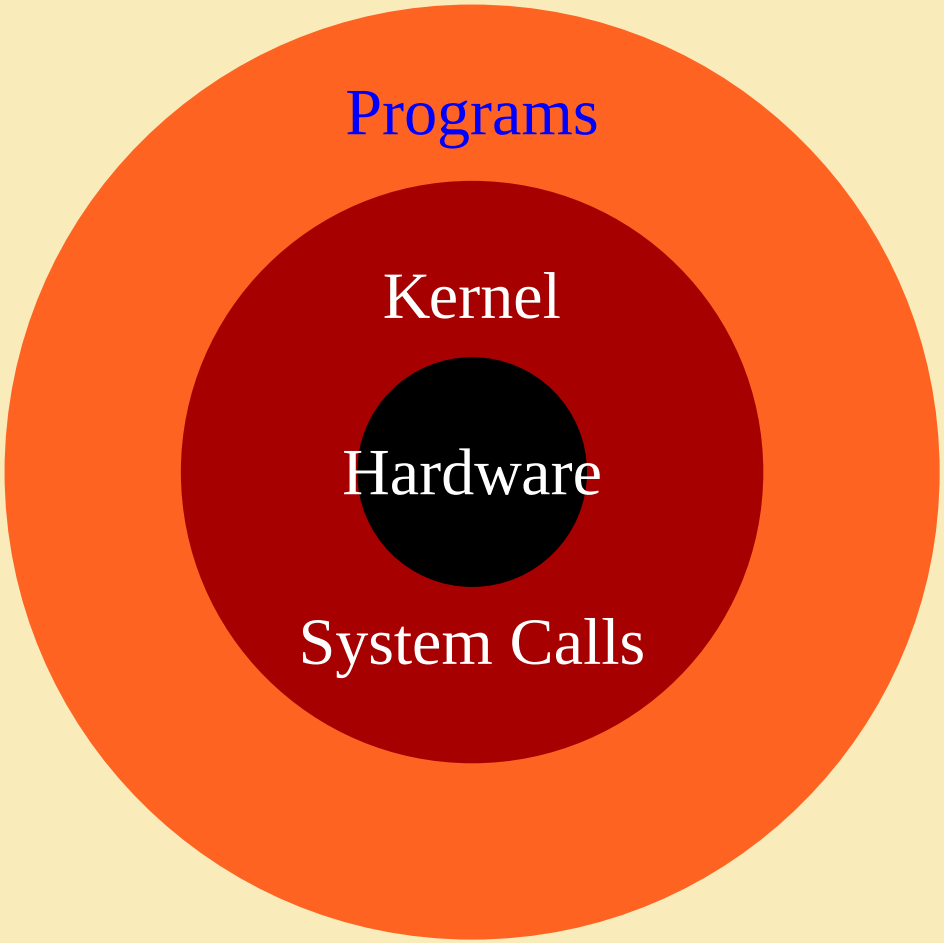
Unix Philosophy

- Multiuser / Multitasking
- Toolbox approach
- Flexibility / Freedom
- Conciseness
- Everything is a file
- File system has places, processes have life
- Designed by programmers for programmers

Unix Structure

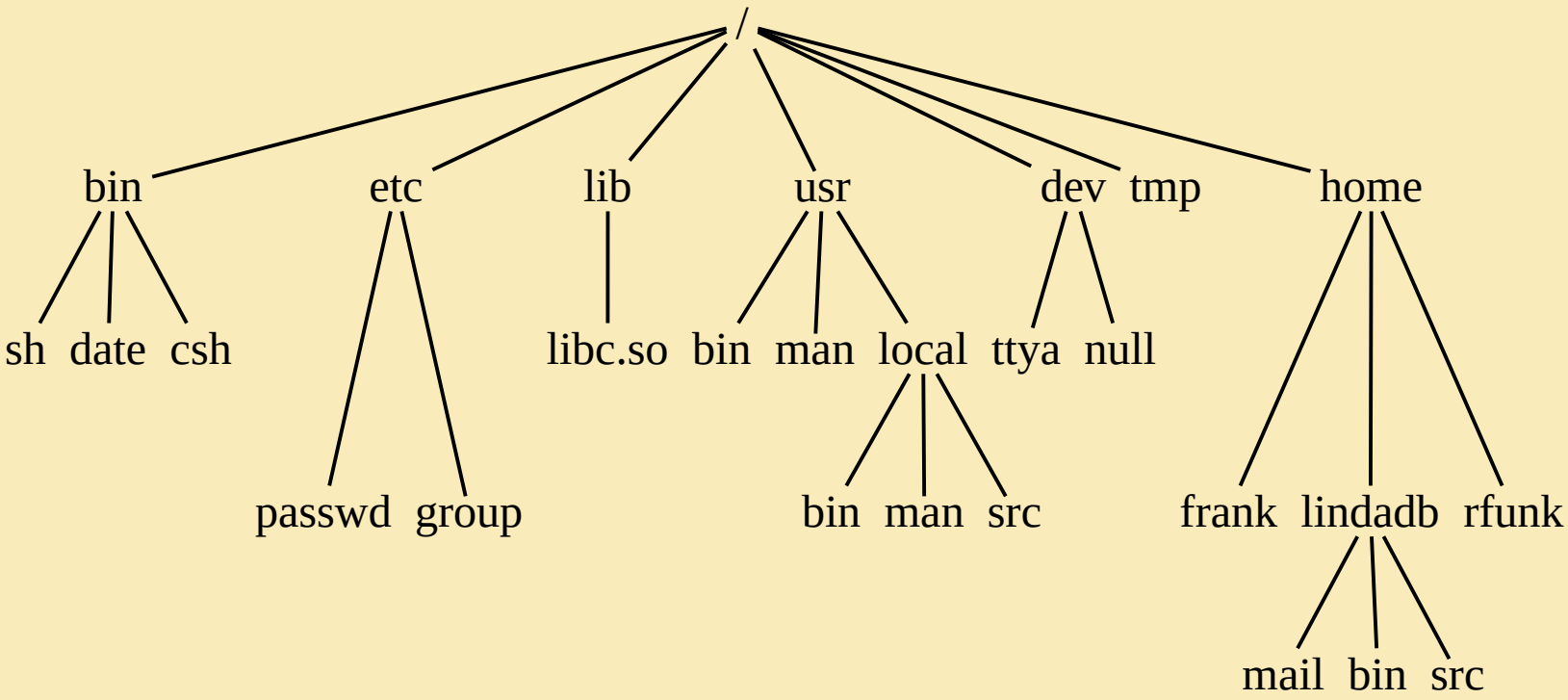
The Operating System

9–10



The File System

11–12



Unix Programs

- Shell is the command line interpreter
- Shell is just another program

A program or command

- interacts with the kernel
- may be any of:
  - built-in shell command
  - interpreted script
  - compiled object code file

Any Questions?

Getting Started — Logging In

- Login and password prompt to log in
- login is user’s unique name
- password is changeable; known only to user, not to system staff
- Unix is case sensitive
- issued login and password (usually in lower case)

Terminal Type

- Default is often to prompt the user
- e.g. vt100, xterm or sun
- To reset:
- `setenv TERM` *terminaltype* (C-shell)
- may need to `unsetenv TERMCAP`
- `TERM=`*terminaltype*`; export TERM` (Bourne shell)

Passwords

Do:

- make sure nobody is looking over your shoulder when you are entering your password.
- change your password often
- choose a password you can remember
- use eight characters, more on some systems
- use a mixture of character types – include punctuation and other symbols

Passwords

15

Don’t:

- use a word (or words) in any language
- use a proper name
- use information in your wallet
- use information commonly known about you
- use control characters
- write your password anywhere
- EVER give your password to anybody

Passwords

15

Your password is your account security:

- To change your password, use the passwd command
- Change your initial password immediately

Exiting

^C	interrupt
^D	can log a user off; frequently disabled
logout	leave the system
exit	leave the shell

Unix Command Line Structure

A command is a program that tells the Unix system to do something. It has the form:

`command` ***options arguments***

- "Whitespace" separates parts of the command line
- An argument indicates on what the command is to perform its action
- An option modifies the command, usually starts with "-"

Unix Command Line Structure

- Not all Unix commands will follow the same standards
- Options and syntax for a command are listed in the “man page” for the command

Getting Help

man: On-Line manual

```
% man command
% man -k keyword
```

Control Keys

Perform special functions

^S	pause display
^Q	restart display
^C	cancel operation
^U	cancel line
^D	signal end of file
^V	treat following control character as normal character

stty - Terminal Control

17–18

- reports or sets terminal control options
- configures aspects of I/O control
- syntax:

```
stty attribute value
```

- example:

```
stty erase ^H
```

Directory Navigation and Control Commands

20–22

pwd	print working directory
cd	change working directory
	(“go to” directory)
mkdir	make a directory
rmdir	remove directory

List directory contents

23–24

ls *[options] [argument]*

-a	list all files
-d	list directory itself, not contents
-l	long listing (lists mode, link info, owner, size, last modification
-g	unix group (requires -l option)

List directory contents

23–24

Each line (when using -l option of ls) includes the following:

- type field (first character)
- access permissions (characters 2–10):
  - first 3: user/owner
  - second 3: assigned unix group
  - last 3: others

Permissions

Permissions are designated:

r	read permission
w	write permission
x	execute permission
-	no permission

s and t also seen in special cases

File Maintenance Commands

25–28

chmod	change the file or directory access permissions (mode)
chgrp	change the group of the file
chown	change the owner of a file

Change permissions on file

27–28

chmod *[options]* file

Using + and - with a single letter:

u	user owning file
g	those in assigned group
o	others

Change permissions on file

27–29

chmod *[options]* file

`chmod u+w file`

gives the user (owner) write permission

`chmod g+r file`

gives the group read permission

`chmod o-x file`

removes execute permission for others

Change permissions on file

27–29

chmod *[options]* file

using numeric representations for permissions:

r	=	4
w	=	2
x	=	1
Total:		7

Change permissions on file

27–29

chmod *[options]* file

chmod 7 7 7 *filename*

user group others

gives user, group, and others r, w, x permissions

Change permissions on file

27–29

chmod *[options]* file

`chmod 750` ***filename***

- gives the user read, write, execute
- gives group members read, execute
- gives others no permissions

Change permissions on file

27–29

chmod *[options]* file

`chmod 640` ***filename***

- gives the user read, write
- gives group members read
- gives others no permissions

Setting default permissions

umask *mask*

- set in startup files for the account
- masks out permissions
- umask numbers added to desired permission number equals 7

File maintenance commands

25–29

chgrp	change the group of the file
	can be done only by member of group
chown	change the ownership of a file
	usually need root access
rm	remove (delete) a file
cp	copy file
mv	move (or rename) file

Display Commands

30–32

echo	echo the text string to stdout
cat	concatenate (list)
head	display first 10 (or #) lines of file
tail	display last 10 (or #) lines of file

more

less page through a text file

pg

Any Questions?

System Resources

33–40

These commands report or manage system resources

Disk space commands

33–35

df *[options] [directory]*

% **df -k /**

du *[options] [directory]* % **du**

% **du** ***directory***

% **du -s** ***directory***

% **du -k** ***directory***

Show status of processes

35–36

ps *[options]*

```
% ps
% ps -ef
% ps auxw
```

Options vary from system to system — see the man pages

Terminate a process

kill *[-signal] processID*

% **kill -l**

displays the available kill signals

% **kill -9** ***processID***

last resort — “nuke” without mercy

User listing

who *[am i]*

% **who**

lists all users currently on system

% **who am i**

reports information on command user

% **whoami**

reports username of command user

Report program locations

37–38

whereis *[options] command*

-b	report binary files only
-m	report manual page files only
-s	report source files only

Examples:

```
% whereis mail
% whereis -b mail
% whereis -m mail
```

Report the command found

`which` ***command***

will report the name of the file that will be executed when the command is invoked

- full path name
- alias found first

Report the name of machine

hostname

reports the name of the machine the user is logged into

uname ***[options]***

has additional options to print info about system hardware and software

Record your session

38–39

script *[-a] [filename]*

-a appends content to a file

```
% script
(...commands...)
% exit
% cat typescript
```

typescript is the default name of the file used by script

Date

date *[options] [+format]*

-u use Universal Time (GMT)

+format:

+%a +%t +%D +%y +%j

```
% date
% date -u
% date +%a%t%D
% date '+%Y:%j'
```

Printing Commands

41–42

BSD and others:

`lpr` ***[options]*** **`filename`**

`lpq` ***[options] [job#] [username]***

`lprm` ***[options] [job#] [username]***

System V:

`lp` ***[options]*** **`filename`**

`lpstat` ***[options]***

`cancel` ***[requestID] [printer]***

Any Questions?

Handy file commands

70

More fun with files

touch — Create a file

touch *[options]* *file*

Options:

-c don’t create file if it doesn’t already exist

```
% touch file
```

ln — Link to another file

ln *[options] source target*

```
% ln -s chkit chkmag
```

symbolic link

```
% ln chkit chkmag2
```

hard link

find — Find files

89–90

find *directory [options] [actions] [...]*

```
% find .  -name ay -ls
% find .  -newer empty -print
% find /usr/local -type d -print
```

Compression

91-93

compress *[options] [file]*

zcat *[file.Z]*

uncompress *[options] [file.Z]*

```
% compress logins.*
% zcat beauty.Z | head
% uncompress logins.*.Z
```

gzip / gunzip often used too – .gz extension

tar — Archive files

tar *[options] [directory/file]*

Options:

-c	create an archive
-t	table of contents list
-x	extract from archive
-f *file*	archive file is named *file*
-v	verbose

tar — Archive files


```
% tar -cf logfile.tar logs.*
% tar -tf logfile.tar
% tar -xf logfile.tar
```

fgrep — find text in a file

fgrep *[options] text [files...]*

The fgrep utility is a simplified version of the grep utility.

fgrep is used to search for exact strings in text files.

Some options for fgrep are:

-i	ignore case
-v	display only lines that dont match
-n	display line number with the line where match was found

Any Questions?

Shells

The shell sits between you and the operating system

- acts as a command interpreter
- reads input
- translates commands into actions to be taken by the system

To see what your current login shell is:

```
echo $SHELL
```

Bourne Shell (sh)

- good features for I/O control — often used for scripts
- not well suited for interactive users
- other shells based on Bourne may be suited for interactive users
- default prompt is $

C Shell (csh)

- uses C-like syntax for scripting
- I/O more awkward than Bourne shell
- nicer for interactive use
- job control
- history
- default prompt is %

- uses ~ symbol to indicate a home directory (user's or others')

Other Shells

Based on the Bourne Shell:

- Korn (ksh)
- Bourne-Again Shell (bash)
- Z Shell (zsh)

Based on the C Shell:

- T-C shell (tcsh)

Built-in Shell Commands

46–47

The shells have a number of built-in commands:

- executed directly by the shell
- don’t have to call another program to be run
- different for the different shells

Environment Variables

DISPLAY

EDITOR

PAGER

PATH

TERM

csh `setenv NAME value`

sh `NAME=value; export NAME`

Shell Variables

PS1 (sh)

prompt (csh)

others as needed

csh **`set name=`*value***

sh **`name=`*value***

These are used by the shell and shell scripts; not seen or used by external programs

Shell startup

The file .profile (sh) or .login (csh) is used at login to:

- set path
- define functions
- set terminal parameters (stty)
- set terminal type
- set default file permissions (umask)

Sample .profile file


```
PATH=/usr/bin:/usr/ucb:/usr/local/bin:.
export PATH
PS1="{ ‘hostname‘ ‘whoami‘ } "
ls() { /bin/ls -sbF "$@"; }
ll() { ls -al "$@"; }
stty erase ^H
eval ‘tset -Q -s -m ’:?xterm’‘
umask 077
```

C Shell Features

50–51

- noclobber
- ignoreeof
- history
- alias

.login and .cshrc

50–51

- .login runs only at login time
- tell whether you have mail
- tell who else is online
- configure terminal settings
- .cshrc runs whenever the shell starts
- set environment and shell variables
- set aliases

Sample .login file


```
# .login
stty erase ^H
set noglob
eval ‘tset -Q -s -m ’:?xterm’ ‘
unset noglob
```

Sample .cshrc file

50–51

```
set path=(/usr/bin /usr/ucb /usr/local/bin ~/bin .)
set prompt = "{‘hostname‘ ‘whoami‘ !}"
set noclobber
set ignoreeof
set history=100 savehist=50
```

Sample .cshrc file

50–51

```
#aliases
alias h history
alias ls "/usr/bin/ls -sbF"
alias ll ls -al
alias cd ’cd \!*;pwd’
umask 077
```

csh Job Control

- Putting a job into the background
- appending & to the command line
- ^Z to stop while job is running
- bg to continue stopped job in background
- fg to return the job to the foreground

csh Job Control

- builtin jobs command to list background jobs
- kill command to kill a background job

History

52–53

C Shell, Korn shell and others retain information about former commands executed within the shell

- Use history and savehist variables to set number of commands retained:
- in .cshrc:

```
set history=100 savehist=50
```

- saved in ~/.history between logins

History shortcuts in csh

`% history` ***nn***

prints last *nn* commands

`% !!`

repeats the last command

`% !`***nn***

repeats the command numbered *nn*

`% !`***string***

repeats latest command starting with *string*

Changing your Shell

- chsh
- passwd -e /usr/local/bin/tcsh

The new shell must be the full path name for the shell on the system

Frequently standard shells:

Bourne:	/bin/sh
Korn:	/bin/ksh
C:	/bin/csh

Changing your Shell

- Alternate shells should be listed in /etc/shells
- tcsh and bash most common alternatives
- Less frustrating to fix typos or redo previous commands.

To try the shell without changing to it, just type its name at your system prompt. (Type exit to return to normal.)

Any Questions?

Special Unix Features

I/O redirection and piping

- output redirection to a file
- input redirection from a file
- piping
- output of one command becomes the input of a subsequent command

Standard File Descriptors

stdin	Standard input to the program
stdout	Standard output from the program
stderr	Standard error output

These are not called by name at shell prompt, but are often referenced by these names.

File Descriptors

55

stdin	normally from the keyboard, but can redirect from a file or command
stdout & stderr	normally to the terminal screen, but can redirect either or both to a file or command

File Redirection

55–57

>	redirect standard output to file
	command > outfile
>>	append standard output to file
	command >> outfile
<	input redirection from file
	command < infile
|	pipe output to another command
	command1 | command2

File Redirection (csh)

55–57

>& file	redirect stdout and stderr to *file*
>>& file	append stdout and stderr to *file*
|& command	pipe stdout and stderr to *command*

To redirect stdout and stderr to separate files:

% **(*command* > *outfile*) >& *errfile***

File Redirection (sh)

55–57

2>file	direct stderr to *file*
>file 2>&1	direct both stdout and stderr to *file*
>>file 2>&1	append both stdout and stderr to *file*
2>&1|command	pipe stdout and stderr to *command*

File Redirection (sh)

55–57

To redirect stdout and stderr to two separate files:

```
$ command > outfile 2 > errfile
```

To discard stderr:

```
$ command 2 > /dev/null
```

(/dev/null is a “black hole” for bits)

Other Special Command Symbols

;	command separator
&	run the command in the background
&&	run the following command only if previous command completes successfully
||	run the following command only if previous command did not complete successfully
( )	grouping — commands within parentheses are executed in a subshell

Quoting

\	escape the following character (take it literally)
' '	don't allow any special meaning to characters within single quotes (except ! in csh)
" "	allow variable and command substitution inside double quotes (does not disable $ and \ within the string)

Backquotes

‘command‘ take the output of *command* and substitute it into the command line

Works inside double-quotes

Wild Cards

?	match any single character
*	match any string of zero or more characters
[abc]	match any one of the enclosed characters
[a-z]	match any character in the range a through z

Wild Cards

[!def] (sh)	match any characters not one of the
[^def] (csh)	enclosed characters
{abc,bcd,cde}	match any set of characters separated by comma (csh)
~	user's own home directory (csh)
~user	home directory of specified user (csh)

Any Questions?

Text Processing

Editors

123–127

Programs that let you interactively edit text files

- Textual
  - vi / vim
  - emacs
  - pico
- Graphical
  - emacs / xemacs
  - dtpad (CDE)
  - jot (SGI)

vi — Visual Editor

- Pronounce both letters: V-I, never “Vy”
- Three modes
  - Command mode (“beep mode”)
  - Insert mode (“no beep mode”)
  - Command line mode (“colon mode”)
- Commands are generally case sensitive

Cursor Movement

126

arrow keys	(depending on terminal)
h, j, k, l	alternates for arrows
[n] h	left *[n]* space(s)
[n] j	down *[n]* space(s)
[n] k	up *[n]* space(s)
[n] l	down *[n]* space(s)

Cursor Movement

126

^F	forward one screen
^B	back one screen
^D	down half screen
^U	up half screen

not case sensitive

Cursor Movement

G	go to last line of file
[n] G	go to last line or line *[n]*
$	end of current line
^	beginning of text on current line
0	beginning of current line
[n] w	forward *[n]* word(s)
[n] b	back *[n]* word(s)
e	end of word

Inserting Text

i	insert text before the cursor
a	append text after the cursor
I	insert text at beginning of line
A	append text at end of line
o	open new line after current line
O	open new line before current line

Deleting Text

dd	delete current line
[n] dd	delete *[n]* line(s)
[n] dw	delete *[n]* word(s)
D	delete from cursor to end of line
x	delete current character
[n] x	delete *[n]* characters
X	delete previous character (like backspace)

Change commands

126

cw	change word
*[n]*cw	change next *[n]* word(s)
c$	change from cursor to end of line
~	change case of character
J	joins current line and next line
u	undo the last command just done

Change commands

.	repeat last change
[n] yy	yank *[n]* line(s) to buffer
[n] yw	yank *[n]* word(s) to buffer
p	puts yanked or deleted text after cursor
P	puts yanked or deleted text before cursor

File Manipulation

:w	write changes to file
:wq	write changes and quit
:w!	force overwrite of file
:q	quit if no changes made
:q!	quit without saving changes
:!	shell escape
:r!	insert result of shell command at cursor position

Any Questions?

Text Processing Commands

- grep / egrep / fgrep — search the argument for all occurences of the search string; list them
- sed — stream editor for editing files from script or command line
- awk / nawk — scan for patterns in a file and process the results

grep

61–64

grep *[options] regexp [files...]*

The grep utility is used to search for regular expressions in Unix files.

fgrep searches for exact strings. egrep uses “extended” regular expressions.

grep options

61–64

Some options for grep are:

- -i ignore case
- -v display only lines that dont match
- -n display line number with the line where match was found

```
grep 'regexp' file
```

Regular Expression Syntax

59–60

Regular expressions:

- allow pattern matching on text
- combine normal and special characters (metacharacters)
- should not be confused with wildcards for matching files

Regular Expression Syntax

59–60

Regular expressions come in three different forms:

- Anchors — tie the pattern to a location on the line
- Character sets — match a single character at a single position
- Modifiers — specify how many times to repeat the previous expression

Regular Expressions can be combined to form longer regular expressions.

Regular Expression Syntax

59–60

.	match any single character except newline
*	match zero or more instances of single expression preceding it
[abc]	match any of the characters enclosed
[a-d]	match any character in enclosed range

Regular Expression Syntax

59–60

[^abc]	match any character NOT in the enclosed set
^exp	regular expression must start at the beginning of the line
exp$	regular expression must end at the end of the line
\	treat the next character literally

Any Questions?

More file processing commands

70

file — File type

86–88

file *[options] file*

Options:

-f *filelist*	*filelist* is a file which contains a list of files to examine
-h	don’t follow symbolic links (SVR4)
-L	follow symbolic links (BSD)

```
% file *
```

strings — find printable strings

strings *[options] file*

Options:

-n *num*	use number as minimum string length
-*num*	(same)
-a	look at all of object file

sort — Sort file contents

79–81

sort *[options] [+pos] file*

-n numeric order

-u unique; omit multiple copies

-f fold upper case to lower case

-d dictionary order (ignore punctuation)

-b ignore leading blanks

uniq — remove duplicate lines

uniq *[options] file [file.new]*

Options:

-d one copy of only the repeated lines

-u select only the lines not repeated

-c include count of duplications in original

```
% uniq file file.new
% uniq -2 file
```

wc — (Word) Count

wc *[options]* *file*

Options:

-c	count bytes
-m	count characters (SVR4)
-l	count lines
-w	count words

```
% wc userlist
```

Any Questions?

Shell Scripts

103

- Similar to DOS batch files
- Quick and simple programming
- Text file interpreted by shell, effectively new command
- List of shell commands to be run sequentially
- Execute permissions, no special extension necessary

Magic first line

103

#!

Include full path to interpreter (shell)

- **`#!/bin/sh`**

- **`#!/bin/csh -f`**

Special Variables (sh)

$#	Number of arguments on command line
$0	Name that script was called as
$1 – $9	Command line arguments
$@	All arguments (separately quoted)
$*	All arguments
$?	Numeric result code of previous command
$$	Process ID of this running script

Interacting With User

110

echo ***output text***

Talk to user (or ask questions)

read ***variable***

Get input from user, put it in variable

Decisions

- test and [
- if [. . .]; then
  . . .
  fi
- case $*variable* in . . . esac
- for *variable* in . . .
  do . . . done

Check sh man page for details, also look at examples.

Any Questions?

Introduction to Unix

Rob Funk

<funk+@osu.edu>

University Technology Services

Workstation Support

http://wks.uts.ohio-state.edu/