CS 241: Systems Programming Lecture 3. More Shell Spring 2020 Prof. Stephen Checkoway

(command) (options) (arguments)

2

(command) (options) (arguments)

(command) is the name of a command or a path to a program

(command) (options) (arguments)

- (command) is the name of a command or a path to a program

(options) are directives to the command to control its behavior

(command) (options) (arguments)

- (command) is the name of a command or a path to a program
- - Short options are a hyphen and a letter: -h

(options) are directives to the command to control its behavior

- (command) is the name of a command or a path to a program
- options are directives to the command to control its behavior

 - Long options are (usually) two hyphens and multiple letters: --help

- (command) is the name of a command or a path to a program
- options are directives to the command to control its behavior

 - Long options are (usually) two hyphens and multiple letters: --help
 - Multiple short options can be combined -a -b -c is the same as -abc



- (command) is the name of a command or a path to a program
- <options
 are directives to the command to control its behavior

 - Long options are (usually) two hyphens and multiple letters: --help
 - Multiple short options can be combined -a -b -c is the same as -abc
 - Options can take arguments: -o file.txt or --output=file.txt



- (command) is the name of a command or a path to a program
- options are directives to the command to control its behavior

 - Long options are (usually) two hyphens and multiple letters: --help
 - Multiple short options can be combined -a -b -c is the same as -abc
- Options can take arguments: -o file.txt or --output=file.txt <arguments are the things the command acts on



- (command) is the name of a command or a path to a program
- <options
 are directives to the command to control its behavior
 - Short options are a hyphen and a letter: -h
 - Long options are (usually) two hyphens and multiple letters: --help
 - Multiple short options can be combined -a -b -c is the same as -abc
 - Options can take arguments: -o file.txt or --output=file.txt
- <arguments are the things the command acts on
 - Often file paths or server names or URLs



- (command) is the name of a command or a path to a program
- <options
 are directives to the command to control its behavior
 - Short options are a hyphen and a letter: -h
 - Long options are (usually) two hyphens and multiple letters: --help
 - Multiple short options can be combined -a -b -c is the same as -abc
 - Options can take arguments: -o file.txt or --output=file.txt
- <arguments are the things the command acts on
 - Often file paths or server names or URLs
 - When no arguments are given (or a single –), many commands read stdin



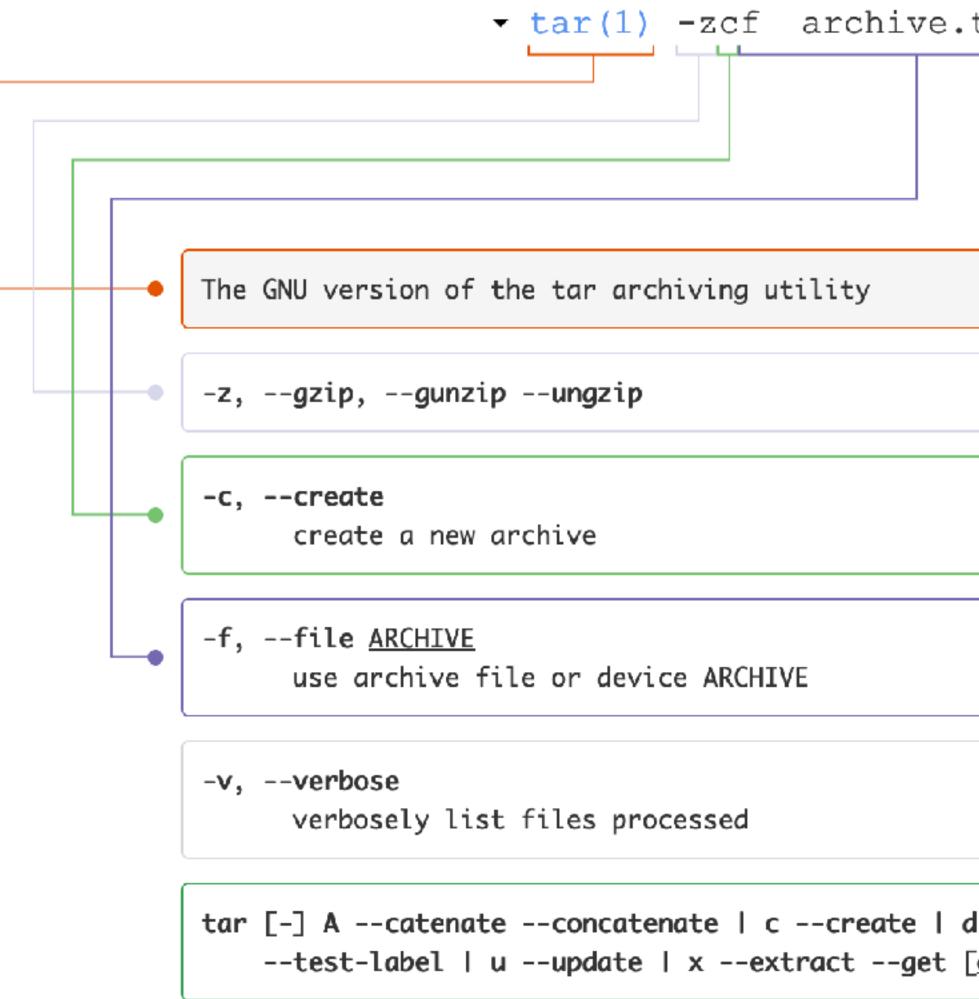
(command) (options) (arguments)

- (command) is the name of a command or a path to a program
- <options
 are directives to the command to control its behavior
 - Short options are a hyphen and a letter: -h
 - Long options are (usually) two hyphens and multiple letters: --help
 - Multiple short options can be combined -a -b -c is the same as -abc
 - Options can take arguments: -o file.txt or --output=file.txt
- <arguments are the things the command acts on
 - Often file paths or server names or URLs
 - When no arguments are given (or a single –), many commands read stdin

Example: tar -zcf archive.tar.gz --verbose dir/file1 file2



Example meaning



Click to go to explainshell.com

tar.gz	verbose	dir/file1	file2		
diffcompare delete rappend tlist [options] [pathname]					
					, ,

3

Shell builtins

- Functionality built into bash (all listed in the manual)
- E.g., cd, alias, echo, pwd

Shell builtins

- Functionality built into bash (all listed in the manual)
- E.g., cd, alias, echo, pwd

Shell functions

User-defined functions (we'll get to these later)

Shell builtins

- Functionality built into bash (all listed in the manual)
- E.g., cd, alias, echo, pwd

Shell functions

User-defined functions (we'll get to these later)

Aliases

E.g., alias ls='ls --color=auto'

Shell builtins

- Functionality built into bash (all listed in the manual)
- E.g., cd, alias, echo, pwd

Shell functions

User-defined functions (we'll get to these later)

Aliases

E.g., alias ls='ls --color=auto'

Programs stored on the file system

- /bin, /usr/bin, /usr/local/bin, /sbin, /usr/sbin
- E.g., ssh, cat, ls, rm

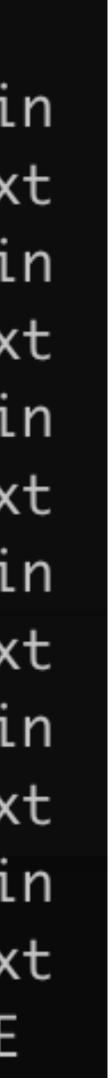
Pathname expansion/globbing

Bash performs pathname expansion via pattern matching (a.k.a. globbing) on each unquoted word containing a wild card

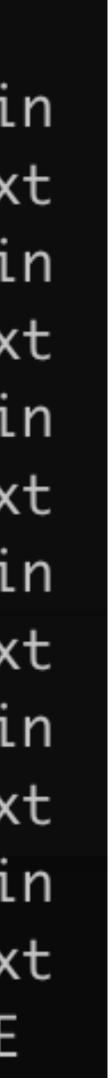
Wild cards: *, ?, [

- * matches zero or more characters
- ? matches any one character
- [!...] or [^...] matches any character not between the brackets
- $\left[x-y \right]$ matches any character in the range, e.g., $\left[a-f \right]$

[...] matches any single character between the brackets, e.g., [abc]



\$ ls ex/*.txt



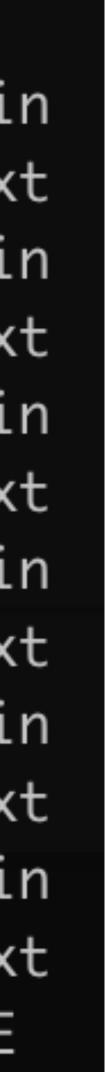
\$ ls ex/*.txt

- ex/a-1.txt ex/a-2.txt
- ex/b-2.txtex/b-3.txt

ex a-1.bin a-1.txt a-2.bin a-2.txt a-3.bin a-3.txt b-1.bin b-1.txt b-2.bin b-2.txt b-3.bin b-3.txt

README

ex/b-1.txt ex/a-3.txt



\$ ls ex/*.txt

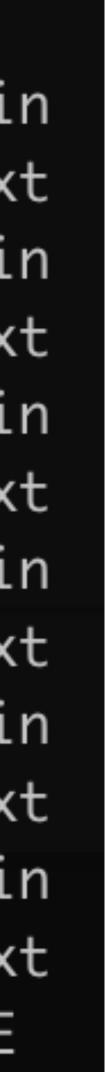
ex/a-1.txt ex/a-2.txtex/b-2.txtex/b-3.txt

\$ ls ex/?-3.*

ex a-1.bin a-1.txt a-2.bin a-2.txt a-3.bin a-3.txt b-1.bin b-1.txt b-2.bin b-2.txt b-3.bin b-3.txt

README

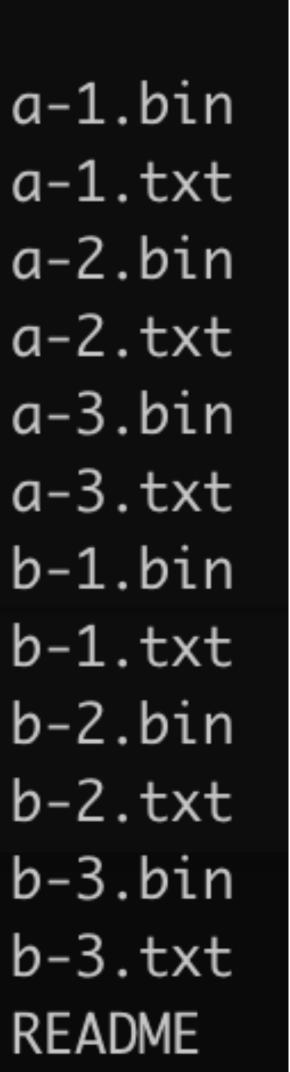
ex/a-3.txtex/b-1.txt



\$ ls ex/*.txt

ex/a-1.txt ex/a-2.txt ex/a-3.txt ex/b-1.txt ex/b-2.txt ex/b-3.txt

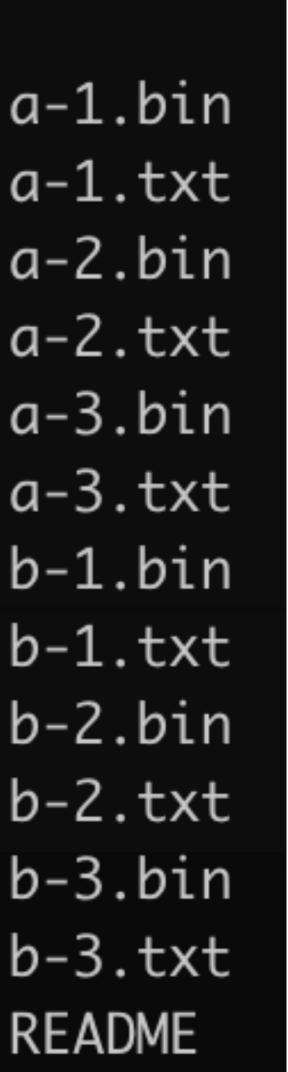
\$ ls ex/?-3.* ex/a-3.bin ex/a-3.txt ex/b-3.bin ex/b-3.txt



\$ ls ex/*.txt ex/a-1.txt ex/a-2.txt ex/a-3.txt ex/b-1.txt ex/b-2.txt ex/b-3.txt

\$ ls ex/?-3.* ex/a-3.bin ex/a-3.txt ex/b-3.bin ex/b-3.txt

\$ ls ex/[^acd]-[0-9].b*in

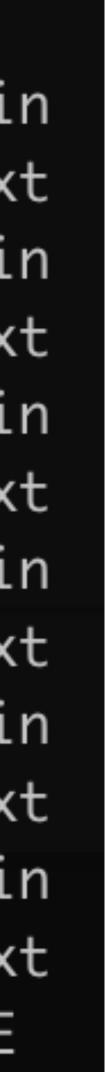


\$ ls ex/*.txt ex/a-1.txt ex/b-2.txt ex/b-3.txt

\$ ls ex/?-3.* ex/a-3.bin ex/a-3.txt ex/b-3.bin ex/b-3.txt

\$ ls ex/[^acd]-[0-9].b*in ex/b-1.bin ex/b-2.bin ex/b-3.bin

ex/a-2.txt ex/a-3.txt ex/b-1.txt

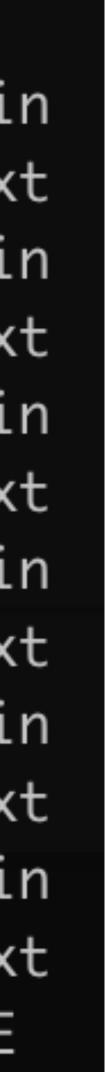


\$ ls ex/*.txt ex/a-1.txt ex/a-2.txt ex/a-3.txt ex/b-1.txt ex/b-2.txt ex/b-3.txt

\$ ls ex/?-3.* ex/a-3.bin ex/a-3.txt ex/b-3.bin ex/b-3.txt

\$ ls ex/[^acd]-[0-9].b*in ex/b-1.bin ex/b-2.bin ex/b-3.bin

ls "ex/*"S

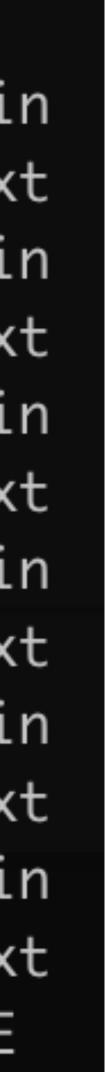


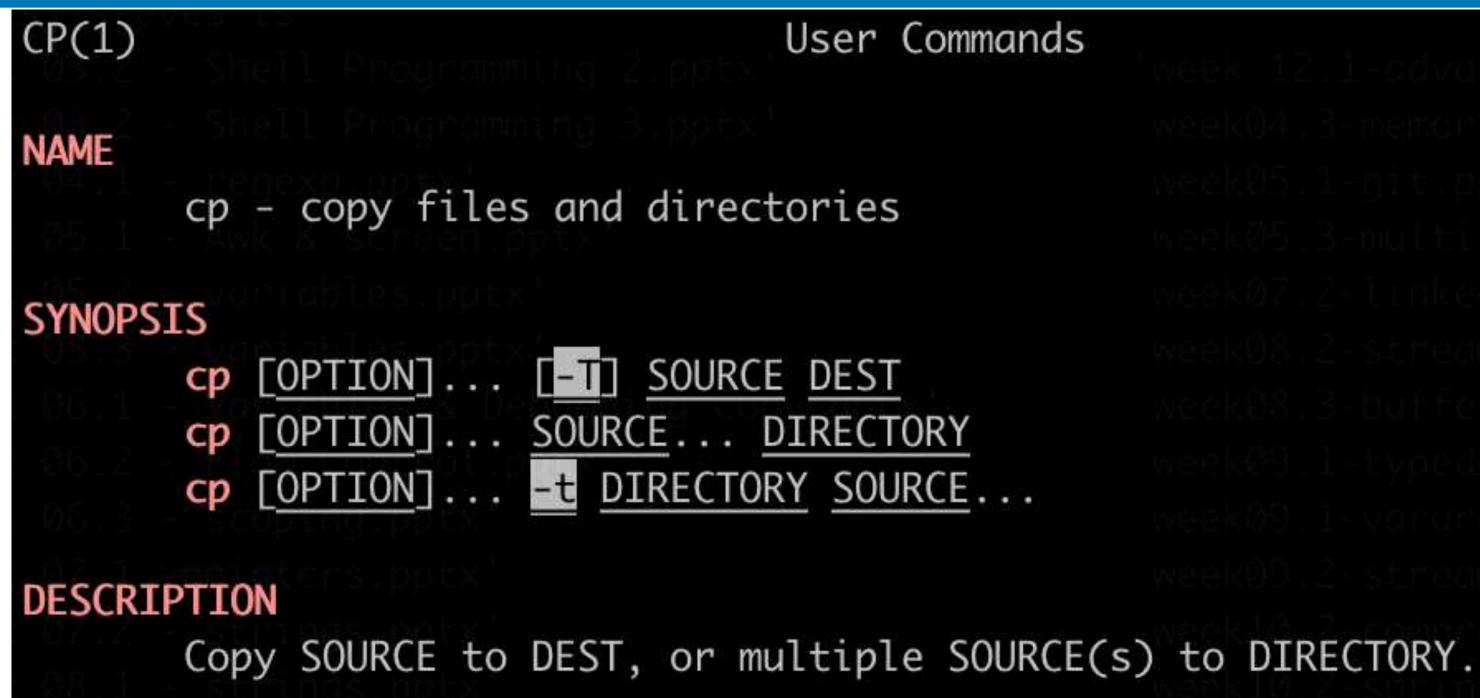
\$ ls ex/*.txt ex/a-1.txt ex/a-2.txt ex/a-3.txt ex/b-1.txt ex/b-2.txt ex/b-3.txt

\$ ls ex/?-3.* ex/a-3.bin ex/a-3.txt ex/b-3.bin ex/b-3.txt

\$ ls ex/[^acd]-[0-9].b*in ex/b-1.bin ex/b-2.bin ex/b-3.bin

⊥s "ex/*" ls: cannot access 'ex/*': No such file or directory





Which command copies all Java source files (those whose names end in java) from the directory a/b to the directory /tmp?

- A. \$ cp a/b/[a-z].java /tmp D. \$ cp a/b/?.java /tmp
- B.\$ cp a/*/*.java /tmp

C.\$ cp a/b/*.java /tmp

r Commands	CP(1)
	week04.3-memory.pptx
0.5	week05.1-git.pptx
es	week05.3-multi-d-arrays.pptx
	week07.2-linked.lists.pptx
ст	week08.2-streams.pptx
<u>ST</u> CTORY	week08.3-buffer.overflöws.pptx
OURCE	week09.1-typedef-bitfields.pptx
UURCE	week09,1-vararigs.pptx
	week09.2-streams.pptx

E.\$ cp a/b /tmp *.java

Typical Unix tool behavior

- \$ program
 - reads from stdin, writes to stdout
- \$ program file1 file2 file3 runs 'program' on the 3 files, write to stdout
- \$ program -
 - For programs that require filenames, might read from stdin

Every running program has (by default) 3 open "files" referred to by their file descriptor number

Input comes from stdin (file descriptor 0)

- input() # Python: Read a line
- \blacktriangleright \$ IFS= read -r var # Read a line and store in var variable

System.in.read(var) // Java: Read bytes and store in var array

Normal output goes to stdout (file descriptor 1)

- > print(var) # Python
- System.out.println(var) // Java
- \$ echo "\${var}" # Bash

Normal output goes to stdout (file descriptor 1)

- > print(var) # Python
- System.out.println(var) // Java
- \$ echo "\${var}" # Bash

- Error messages traditionally go to stderr (file descriptor 2) > print(var, file=sys.stderr) # Python System.err.println(var) // Java \$ echo "\${var}" >&2 # Bash

Redirection

>file — redirect standard output (stdout) to file with truncation

>file — redirect standard output (stdout) to file with truncation

>>file — redirect stdout to file, but append

- >file redirect standard output (stdout) to file with truncation
- >>file redirect stdout to file, but append
- <file redirect input (stdin) to come from file</pre>

- >file redirect standard output (stdout) to file with truncation
- >>file redirect stdout to file, but append
- <file redirect input (stdin) to come from file</pre>
- connect stdout from left to stdin on right

- >file redirect standard output (stdout) to file with truncation
- >>file redirect stdout to file, but append
- <file redirect input (stdin) to come from file</pre>
- connect stdout from left to stdin on right
 \$ Is wc

- >file redirect standard output (stdout) to file with truncation
- >>file redirect stdout to file, but append
- <file redirect input (stdin) to come from file</pre>
- connect stdout from left to stdin on right
 \$ Is wc
- 2>file redirect standard error (stderr) to file with truncation

- >file redirect standard output (stdout) to file with truncation
- >>file redirect stdout to file, but append
- <file redirect input (stdin) to come from file</pre>
- connect stdout from left to stdin on right
 \$ Is wc
- 2>file redirect standard error (stderr) to file with truncation
- 2>&1 redirect stderr to stdout

\$ echo 'Hi!' >output.txt

- \$ echo 'Hi!' >output.txt
- \$ cat <input.txt</pre>

- \$ echo 'Hi!' >output.txt
- \$ cat <input.txt</pre>
- \$ sort <input.txt >output.txt

- \$ echo 'Hi!' >output.txt
- \$ cat <input.txt</pre>
- \$ sort <input.txt >output.txt
- \$ ps -ax | grep bash

- \$ echo 'Hi!' >output.txt
- \$ cat <input.txt</pre>
- \$ sort <input.txt >output.txt
- \$ ps -ax | grep bash
- \$ grep hello file | sort | uniq -c

- \$ echo 'Hi!' >output.txt
- \$ cat <input.txt</pre>
- \$ sort <input.txt >output.txt
- \$ ps -ax | grep bash
- \$ grep hello file | sort | uniq -c
- \$ echo Hello | cut -c 1-4 >>result.txt

- \$ echo 'Hi!' >output.txt
- \$ cat <input.txt</pre>
- \$ sort <input.txt >output.txt
- \$ ps -ax | grep bash
- \$ grep hello file | sort | uniq -c
- \$ echo Hello | cut -c 1-4 >>result.txt
- \$./process <input | tail -n 4 >output

(Almost) everything is a file

Files on the file system

Network sockets (for communicating with remote computers, e.g., web browsers, ssh, mail clients etc.)

Terminal I/O

A bunch of special files

- /dev/null
- /dev/zero

— Writes are ignored, reads return end-of-file (EOF) - Writes are ignored, reads return arbitrarily many 0 bytes /dev/urandom — Reads return arbitrarily many (pseudo) random bytes





Given that /dev/null ignores all data written to it, how can we run the program ./foo and redirect stderr so no error messages appear in our terminal?

A.\$./foo >/dev/null
B.\$./foo 1>/dev/null
C.\$./foo 2>/dev/null
D.\$./foo | /dev/null
E.\$./foo &2>/dev/null

a program ./foo such that it has no input at all?

- A.\$./foo </dev/null
- B.\$./foo </dev/zero
- C.\$./foo </dev/urandom
- D.\$./foo </dev/eof</pre>
- E.\$ echo | ./foo

Some programs read all of their input before terminating. How can we run

In-class exercise

Grab a laptop and a partner and try to get as much of that done as you can!



https://checkoway.net/teaching/cs241/2020-spring/exercises/Lecture-03.html