EECS2031 Software Tools

S 2020-21

Nov 26/29, 2021 Lecture 22

90

Utilities II – advanced utilities

Introduces utilities for power users, grouped into logical sets

We introduce about thirty useful utilities.

Section	Utilities
Filtering files	egrep, fgrep, grep, uniq
Sorting files	sort
Extracting fields	cut
Comparing files	cmp, diff
Archiving files	tar, cpio, dump
Searching for files	find
Scheduling commands	at, cron, crontab
Programmable text processing	awk, perl
Hard and soft links	ln
Switching users	su
Checking for mail	biff
Transforming files	compress, crypt, gunzip, gzip, sed, tr, ul, uncompress
Looking at raw file contents	od
Mounting file systems	mount, umount
Identifying shells	whoami
Document preparation	nroff, spell, style, troff
Timing execution of commands	time

91

91

Removing Duplicate Lines: uniq

- The uniq utility displays a file with all of its identical adjacent lines replaced by a single occurrence of the repeated line.
- Here's an example of the use of the uniq utility:

```
$ cat  animals          # look at the test file.
cat  snake
monkey snake
dolphin elephant
dolphin elephant
goat elephant
pig pig
pig pig
monkey pig
pig pig
```

```
$ uniq  animals     # filter out duplicate adjacent lines.
cat  snake
monkey snake
dolphin elephant
goat elephant
pig pig
monkey pig
pig pig
```

How to merge non-adjacent lines too? sort and then pipe to uniq

sort

- sorts a file in ascending or descending order based on one or more fields.
- Individual fields are ordered lexicographically, which means that corresponding characters are compared based on their ASCII values.

-t field separator/delimiter (default is blank or tab)
-r descending instead of ascending
-f ignore case

-k key sort on field/column
-n numeric sort
-M month sort (3 letter month abbreviation)

Hello the world
field 3
delimiter

sort examples

```
$ cat  data.txt
John   Smith      1222  26  Apr   1956
Tony   Jones      1012  20  Mar  1950
John   Duncan     2     20  Jan  1966
Larry  Jones      3223  20  Dec 1946
Lisa   Sue        1222  4   Jul   1980

$ sort data.txt        # cat data.txt | sort
John  Duncan      2     20  Jan  1966
John  Smith       1222  26  Apr  1956
Larry Jones       3223  20  Dec 1946
Lisa  Sue         1222  4   Jul  1980
Tony Jones        1012  20  Mar 1950

$ sort -r data.txt    # descending
Tony Jones        1012  20 Mar 1950
Lisa  Sue         1222  4  Jul   1980
Larry Jones       3223  20 Dec 1946
John  Smith       1222  26 Apr  1956
John  Duncan      2     20 Jan  1966
```

Whole lines are ordered lexicographically

94

sort -k -n -k by column -n numerical

```
$ sort -k2   data.txt          #  -k 2
John   Duncan  2       20  Jan 1966
Tony   Jones   1012    20  Mar 1950
Larry  Jones   3223    20  Dec 1946
John   Smith   1222    26  Apr  1956
Lisa   Sue     1222    4   Jul   1980
```

sort by column 2, surname

```
$ sort -k3   data.txt          #  -k 3
Tony  Jones    1012    20  Mar 1950
Lisa  Sue      1222    4   Jul   1980
John  Smith    1222    26  Apr  1956
John  Duncan   2       20  Jan 1966
Larry Jones    3223    20  Dec 1946
```

sort by field/column 3

Lexicographically column 3 not sorted correctly

```
$ sort -k3 -n  data.txt        #  -nk3
John  Duncan   2       20  Jan 1966
Tony  Jones    1012    20  Mar 1950
John  Smith    1222    26  Apr  1956
Lisa  Sue      1222    4   Jul   1980
Larry Jones    3223    20  Dec 1946
```

-n enables column 3 to be sorted numerically

95

sort -k -n

```
$ sort -k3   data.txt          #  -k 3        +2 -3   (start 0)
Tony  Jones     1012    20  Mar 1950
John  Duncan    2       20  Jan 1966
Lisa  Sue       1222    4   Jul  1980
John  Smith     1222    26  Apr 1956
Larry Jones     3223    20  Dec 1946
```

Lexicographically column 3 not sorted correctly

```
$ sort -k3 -n  data.txt        #  -nk3  -nk 3    +2 -3  start 0
John  Duncan    2       20  Jan 1966
Tony  Jones     1012    20  Mar 1950
John  Smith     1222    26  Apr 1956
Lisa  Sue       1222    4   Jul  1980
Larry Jones     3223    20  Dec 1946
```

-n enables column 3 to be sorted numerically

```
$ sort  -k3  -k4  -n data.txt      #  +2 -3 +3 -4
John  Duncan    2       20  Jan  1966
Tony  Jones     1012    20  Mar  1950
Lisa  Sue       1222    4   Jul   1980     # Lisa and John further sorted
John  Smith     1222    26  Apr  1956
Larry Jones     3223    20  Dec 1946
```

For your information

sort -M

```
$ sort -k5 data.txt            #   +4 -5
John  Smith     1222  26   Apr   1956
Larry  Jones    3223  20   Dec   1946
John  Duncan    2     20   Jan   1966
Lisa  Sue       1222  4    Jul   1980
Tony  Jones     1012  20   Mar   1950
```

Lexicographically Months not sorted correctly

sort -M

```
$ sort -k5 data.txt                # +4 -5
John  Smith     1222  26   Apr   1956
Larry  Jones    3223  20   Dec   1946
John  Duncan    2     20   Jan   1966
Lisa  Sue       1222  4    Jul   1980
Tony  Jones     1012  20   Mar   1950
```

Lexicographically Months not sorted correctly

```
$ sort  -k5 -M data.txt
John  Duncan    2     20   Jan   1966
Tony  Jones     1012  20   Mar   1950
John  Smith     1222  2    Apr   1956
Lisa  Sue       1222  46   Jul   1980
Larry Jones     3223  20   Dec   1946
```

-M enables months to be sorted correctly

```
$ sort  -k5 -M -r data.txt
Larry  Jones    3223  20   Dec   1946
Lisa  Sue       1222  46   Jul   1980
John  Smith     1222  26   Apr   1956
Tony  Jones     1012  20   Mar   1950
John  Duncan    2     20   Jan   1966
```

-r reverse descending

sort year?

Two more examples

- who | sort

```
aboelaze pts/20    2019-07-10  16:26 (6.dsl.bell.ca)
farhanieh pts/0    2019-06-26  14:05 (:20)
franck   pts/25    2019-06-30  13:28 (gradchair.eecs.yorku.ca)
franck   pts/6     2019-07-08  07:11  (5.cpe.teksavvy.com)
fwei     pts/10    2019-07-08  11:35  (net.cable.rogers.com)
fwei     pts/14    2019-07-08  11:42  (net.cable.rogers.com)
```

- who | sort -k3

```
farhanieh pts/0    2019-06-26  14:05 (:20)
franck   pts/25    2019-06-30  13:28 (gradchair.eecs.yorku.ca)
franck   pts/6     2019-07-08  07:11  (5.cpe.teksavvy.com)
fwei     pts/10    2019-07-08  11:35  (net.cable.rogers.com)
fwei     pts/14    2019-07-08  11:42  (net.cable.rogers.com)
aboelaze pts/20    2019-07-10  16:26 (6.dsl.bell.ca)
```

- ls -l | sort -k5 -n

99

Get first/last two people?

Let's Do

Two more examples sort -t (default is blank or tab)

- cat /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
```

For your information

- cat /etc/passwd | sort -t : -k4 -n # -t ":" use : as delimiter

```
halt:x:7:0:halt:/sbin:/sbin/halt
operator:x:11:0:operator:/root:/sbin/nologin
root:x:0:0:root:/root:/bin/bash
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
sync:x:5:0:sync:/sbin:/bin/sync
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
```


sort + uniq Merge all identical lines

- uniq is a little limited but we can combine it with sort

```
sort | uniq
```

which are almost the so the first question you

	sort →		uniq →	
cat snake		cat snake		cat snake
monkey snake		monkey snake		monkey snake
dolphin elephant		dolphin elephant		dolphin elephant
dolphin elephant		dolphin elephant		goat elephant
goat elephant		goat elephant		pig pig
pig pig		pig pig		monkey pig
pig pig		pig pig		
monkey pig		pig pig		
pig pig		monkey pig		

Utilities II – advanced utilities

Introduces utilities for power users, grouped into logical sets

We introduce about thirty useful utilities.

Section	Utilities
Filtering files	egrep, fgrep, grep, uniq
Sorting files	sort
Extracting fields	cut
Comparing files	cmp, diff
Archiving files	tar, cpio, dump
Searching for files	find
Scheduling commands	at, cron, crontab
Programmable text processing	awk, perl
Hard and soft links	ln
Switching users	su
Checking for mail	biff
Transforming files	compress, crypt, gunzip, gzip, sed, tr, ul, uncompress
Looking at raw file contents	od
Mounting file systems	mount, umount
Identifying shells	whoami
Document preparation	nroff, spell, style, troff
Timing execution of commands	time

104

104

Comparing Files: cmp, diff

- There are two utilities that allow you to compare the contents of two files:

- cmp, which finds the first byte that differs between two files
- diff, which displays all of the differences and similarities between two files

- Testing for sameness: cmp
  - determines whether two files are the same.
  - produces nothing if the files are identical, exit code 0
  - produce some info otherwise, exit code 1

```
$ cat lady1                          # look at the first test file.
Lady of the night,
I hold you close to me,
And all those loving words you say are right.

$ cat lady2                          # look at the second test file.
Lady of the night,
I hold you close to me,
And everything you say to me is right.

$ cmp lady1  lady2                   # files differ.
lady1 lady2 differ: char 48, line 3
$
```

105

105

File Differences: diff

- The **diff** utility compares two files and displays a list of editing changes that would convert the first file into the second file.
  - produce nothing if the two files are identical, exit code 0
  - produce some info otherwise, exit code 1

```
$ diff lady1 lady2          # compare lady1 and lady2.
3c3
< And all those loving words you say are right.
…
> And everything you say to me is right.
$ _
```

```
$ gcc yourCode;
$ a.out > yourOutput;
$ cmp yourOutput sampleOutput;  # or diff
$ echo $?
```

Exit code $?
0 identical
1 not identical

106

106

cut deal with fields (columns)

-d -f

- Used to split lines of a file
- A line is split into fields
- Fields are separated by delimiters/separators
- A common case where a delimiter is a space:
  - Default is tab, (not " ") need to set it if blank is delimiter
    -d " "
  - cut -f3 -d" "
- **hello there world**

field 3

delimiter

107

107

```
$ cat data.txt          # assuming tab as delimiter
John    Smith    1222    26    Apr  1956
Tony    Jones    1012    20    Mar  1950
John    Duncan   1111    20    Jan  1966
Larry   Jones    1223    20    Dec  1946
Lisa    Sue      1222    15    Jul  1980

$ cut -f 1  data.txt     # show field 1,  tab as delimiter
John
Tony
John
Larry
Lisa

$ cut -f 1,3  data.txt
John    1222
Tony    101
John    1111
Larry   1223
Lisa    1222

$ cut -f 1-3  data.txt
John    Smith    1222
Tony    Jones    101
John    Duncan   1111
Larry   Jones    1223
Lisa    Sue 1222

$ cut -f 1,2  data.txt > names.txt

# classlist example
$ grep -w Wang classlist | cut -f 3,4
$ cut -f 3,4  data.txt | grep -w Wang
```

108

Utilities II – advanced utilities

Introduces utilities for power users, grouped into logical sets

We introduce about thirty useful utilities.

Section	Utilities
Filtering files	egrep, fgrep, grep, uniq
Sorting files	sort
Extracting fields	cut
Comparing files	cmp, diff
Archiving files	tar, cpio, dump
Searching for files	find
Scheduling commands	at, cron, crontab
Programmable text processing	awk, perl
Hard and soft links	ln
Switching users	su
Checking for mail	biff
Transforming files	compress, crypt, gunzip, gzip, sed, tr, ul, uncompress
Looking at raw file contents	od
Mounting file systems	mount, umount
Identifying shells	whoami
Document preparation	nroff, spell, style, troff
Timing execution of commands	time

109

109

find Utility

find pathList expression

- finds files starting at pathList
- finds files descending from there

```
find . -name "lab3a.c"
```

Do it in tryC/21S

- allows you to perform certain actions on results
  - e.g., copying (`cp`), renaming (`mv`), deleting (`rm`) the files

"Find file lab3a.c and rename it to lab3a.bak"

```
find . -name "lab3a.c"  -exec mv {} {}.bak \;
```


find Utility

- -name *pattern*
  True if file's name matches *pattern*, which include shell metacharacters * ? [ ]

  Filename wildcards
- -mtime *count*
  True if the content of the file has been modified within *count* days
- -atime *count*
  True if the file has been accessed within *count* days
- -ctime *count*
  True if the contents of the file have been modified within *count* days or any of its file attributes have been modified
- - type -maxdepth

- -exec *command*
  True if the exit code = 0 from executing the command.
  - *command* must be terminated by \;
  - If {} is specified as a command line argument, it is replaced by the file name currently matched

find examples

- $ find / -name x.c # search for file/dir named x.c in the entire file system
- $ find . -name x.c # search for files/dir named x.c in current and subdirectories
- $ find . -name '*.bak' # "*.bak" search for all bak files in current and subdirectories

  Filename wildcards
- $ find . -name 'a?.c' # "a?.c" search for all file/dir named aX.c

  a1.c
  a2.c
  a3.c

  Filename wildcards
- $ find . -name 'a?.c' | wc -l # how many

112

find examples

- $ find / -type f # search for files (only), in the entire file system
- $ find . -mtime 14 # search for files/dir modified in the last 14 days in current and subdirectories
- $ find . -type f -mtime 14 # search for files (only) modified in the last 14 days, in current and subdirectories
- $ find . -type d # list all directories
- $ find . -type d -name 'lab*' # search for all directories named lab* in current and subdirectories
- $ find . -maxdepth 1 -type f -name 'lab*' # search for all file (only) named lab*, in current directory only (no sub-directories)
- $ find . -type f -name 'a?' | wc -l # how many files (no directory) with name a?, in current and subdirectories

113

For your information

113

find examples -exec

- $ find . -name '*.class' -exec rm { } \; # remove all java class files
  rm ./MyLinkedList.class # possible in Windows?
  rm ./a3/My3.class
  ….
- $ find . -name 'a?.c' -exec cp {} {}.bak \;
  cp a1.c a1.c.bak # find aX.c files and then copy them to aX.c.bak
  cp a2.c a2.c.bak
  ….
- $ find . -name '*.c' -exec mv {} {}.2031 \;
  mv a1.c a1.c.2031 # find all c files and then rename them to
  mv lab3b.c lab3b.c.2031 filename.c.2031
  find . -name '*.c' -exec rm {} \;
  ….
- $ find . -name '*.c' -exec mv {} ../archive/2031 \;
  mv a1.c ../archive/2021W # find all c files and move them to
  directory ../archive/2021W
  ….
- $ find . -name '*.java' -exec chmod 770 {} \;
  chmod 770 MyLL.java # find all java files and change mode to rwxrwx---

114

Utilities II – advanced utilities

Regular Expression

grep/egrep `grep -w -i ^[Tt]he file123`

sort `sort -t : -k 4 -r -n/M file`

default delimiter: blank/tab

cut `cut -d" " -f 2,3 file`

Default delimiter: tab

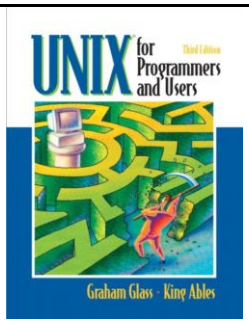
find `find . -name "*.c" -exec cp {} {}.bak \;`

Default: subdirectoies -maxDepth x to limit

find . -type d -exec cp data.txt {} \; // for TA lab6LLX

116

Contents



- Overview of UNIX
  - Structures
  - File systems
    - Pathname: absolute vs relative
    - Security `-rwx-rw--x`
  - Process:
    - Exit code ≥ 0
    - IPC: Pipes

      who | sort who | sort | head -3 who | grep Wang | wc -l
- Utilities/commands
  - Basic: pwd, ls, rmdir, mkdir, cat, more, mv, cp, rm, wc, chmod
  - Advanced: **grep/egrep, sort, cut, find** ….
- **Shell (common shell functionalities)** today
- Bourn (again) Shell
  - scripting language

117

117

- **INTRODUCTION**

A shell is a program that is an interface between a user and the raw operating system.

It makes basic facilities such as multitasking and piping easy to use, and it adds useful file-specific features such as wildcards and I/O redirection.

There are four common shells in use:

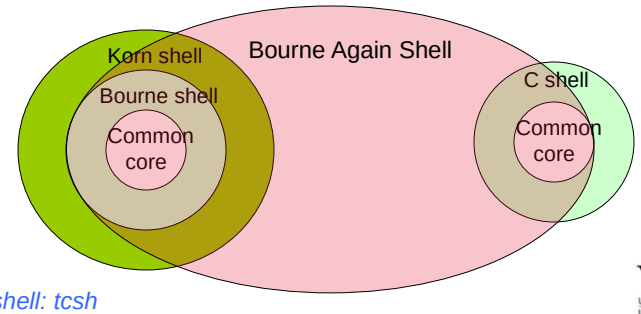
- the Korn shell
- the C shell
- the Bourne shell
- the Bash shell (Bourne Again Shell)

118

SHELL FUNCTIONALITY

- This part describes the common core of functionality that all four shells provide
  - E.g., pipe who | sort
  - E.g., filename wildcards ls *.c ls a?.c

- The relationship among the four shells:



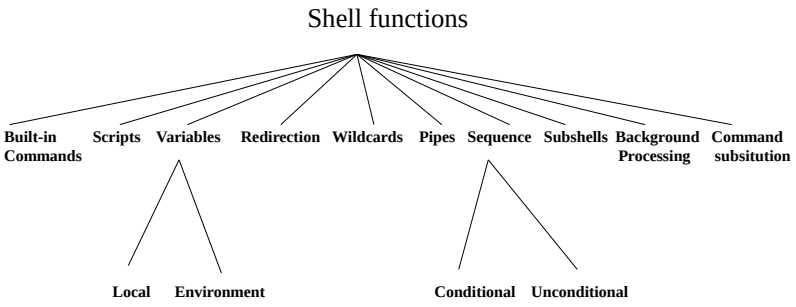
Login shell: tcsh

An enhanced but based on and completely compatible version of the C shell, *csh*

119

SHELL FUNCTIONALITY

A hierarchy diagram to illustrate the features shared by the four shells



120

- **SHELL OPERATIONS**

Commands range from simple utility invocations like:

$ ls

to complex-looking pipeline sequences like:

$ cat xFilecompact123 | sort | uniq | cut -f 2 | head -3

121

- **METACHARACTERS**

Some characters are processed specially by a shell and are known as metacharacters.

All four shells share a core set of common metacharacters, whose meanings are as follow:

Symbol	Meaning
>	Output redirection; writes standard output to a file.
>>	Output redirection; appends standard output to a file.
<	Input redirection; reads standard input from a file.
<<	Input redirection; reads standard input from script up to tok.
*	Filename-substitution (wildcard); matches zero or more characters.
?	Filename-substitution (wildcard); matches any single character.
[...]	Filename-substitution (wildcard); matches any character between the brackets.

Don't confuse with Regex

122

Shell functions

Symbol	Meaning
`command`	Command substitution; replaced by the output from command
$	Variable substitution. Expands the value of a variable.
&	Runs a command in the background. jedit&
|	Pipe symbol; sends the output of one process to the input of another
;	Used to sequence commands. echo hello; wc lyrics
||	Conditional execution; executes a command if the previous one fails.
&&	Conditional execution; executes a command if the previous one succeeds.
(…)	Groups commands.
#	All characters that follow up to a new line are ignored by the shell and program (i.e., used for a comment)
\	Prevents special interpretation of the next character.
' ' " "	quoting
Scripts	

125

- **When you enter a command, the shell scans it for metacharacters and (if any) processes them specially**

  When all metacharacters have been processed, the command is finally executed.

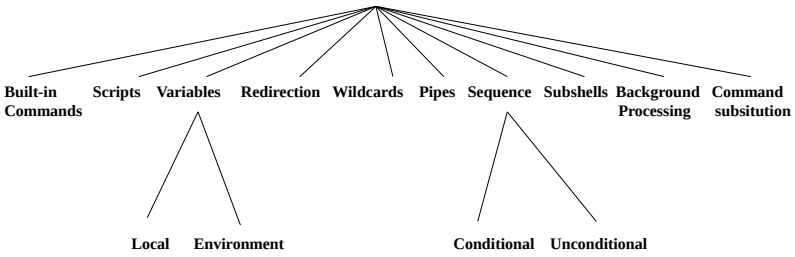
- To turn off the special meaning of a metacharacter, precede it by a backslash(\) character. # Also " " ' ' (later)

- Here's an example:

```
$ echo  hi > file      #  store output of echo in "file".
$ cat  file            #  look at the contents of "file".
hi

$ echo hi \> file      #  inhibit > metacharacter.
hi > file              #  > is treated like other characters.
$ cat file             #  look at the file again.  Not written
ls: cannot access file: No such file or directory such a file

$ echo 3 + 2 = 5
$ echo 3 \* 4 = 12
```

126

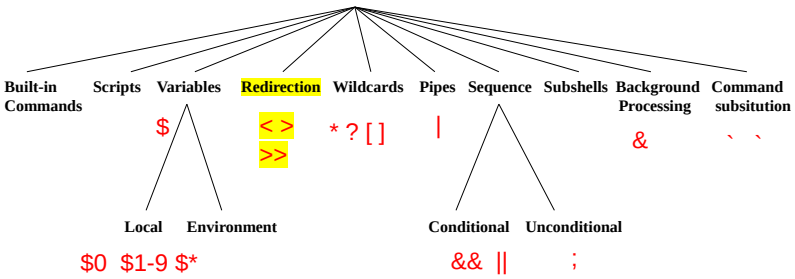
Shell functions



Now, let's go through the functionalities, along with their associated metacharacters

127

Shell functions



- Covered core shell functionality
  - Built-in commands/utilities
  - Redirection < > >>
  - Wildcards (filename substitution) * ? [ ]
  - Pipes |
  - Command substitution ` `
  - Sequence ; conditional sequence && ||
  - Background processing & Grouping ()
  - Variables $ variable substitution
  - Quoting ' ' " "
  - Scripts

128

Shell functions

Built-in Commands | Scripts | Variables (Local, Envirnment) | Redirection | Wildcards | Pipes | Sequence (Conditional, Unconditional) | Subshells | Background Processing | Command subsitution

- **Redirection > >> < <<**

The shell redirection facility allows you to:

1) store the output of a process to a file (output redirection)
2) use the contents of a file as input to a process (input redirection)

Output redirection

To redirect output, use either the > or >> metacharacters.

```
$ a.out > fileName
$ cat file1 file2  > file3
$ cut -f 3,4  classlist > names.txt
```

Difference?

```
$ echo "new line"   >   filename    # create or overwrite filename
$ echo "new line"   >> filename     # append to (end of) filename
```


Input Redirection

Input redirection is useful because it allows you to prepare a process input beforehand and store it in a file for later use.

To redirect input, use either the < or << metacharacters.

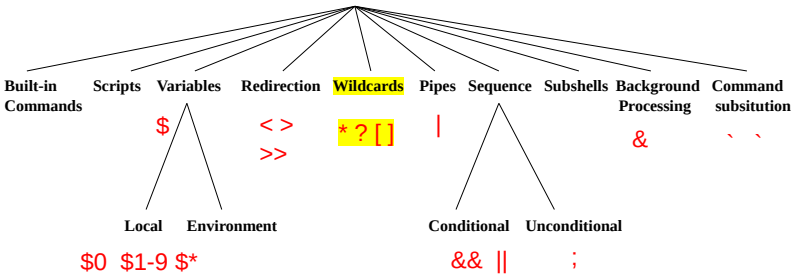
The sequence

```
$ a.out < inputA.txt

$ a.out < ../inputA.txt    # relative path
```

executes command using the contents of the file inputA.txt as its standard input.

If the file doesn't exist or doesn't have read permission, an error occurs.

Shell functions

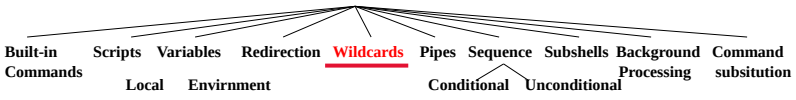


- Covered core shell functionality
  - Built-in commands/utilities
  - Redirection < > >>
  - Wildcards (filename substitution) * ? [ ]
  - Pipes |
  - Command substitution ` `
  - Sequence ; conditional sequence && ||
  - Background processing & Grouping ()
  - Variables $ variable substitution
  - Quoting ' ' " "
  - Scripts

YORK UNIVERSITÉ UNIVERSITY

131

Shell functions



- **FILENAME SUBSTITUTION (WILDCARDS)**

All shells support a wildcard facility that allows you to select files that satisfy a particular name pattern from the file system.

The wildcards and their meanings are as follows:

Wildcard	Meaning	
*	Matches any string, including the empty string.	ls *.c
?	Matches any single character. (Exact one)	ls a?.c
[..]	Matches any one of the characters between the brackets. A range of characters may be specified by separating a pair of characters by a hyphen. [ab] [a-d] [0-9]	

Don't confuse with Regulation Expression → grep a*b lab2a.c | grep a?c file123

132

Used for filename wildcard, in ls, cp, mv, rm, cat, more, wc, chmod grep, find -name operating on multiple files

Here are some examples of wildcards in action:

```
$ ls   *.c       # list files whose name ends  in ".c ".  I.e., list all C files
a.c      b.c      ax.c

$ ls   ?.c       # files whose name is exactly one character followed by ".c"
a.c      b.c

$ ls  a*.c       # a following by anything (including empty)  before .c
a.c      ax.c

$ ls  a?.c       # a followed by exactly one character before .c
ax.c
```

```
$ cp /eecs/dept/course/2019-20/W/2030tmp/xFile?   ./
$ cp /eecs/dept/course/2019-20/W/2030tmp/xFile*   ./
$ cp /eecs/dept/course/2019-20/W/2030tmp/xFile[23]  .

$ submit setArr[1-3].c   $ submit setArr[123].c    $ submit setArr?.c
```

133

Used for filename wildcard, in ls, cp, mv, rm, cat, more, wc, chmod grep, find -name operating on multiple files

```
$ ls  *.c       # list files whose name ends  in ".c ".  I.e., list all C files
a.c      b.c      ax.c     axb.c    axy.c     axyz.c  ?

$ ls  ?.c       # files whose name is exactly one character followed by ".c"
a.c      b.c

$ ls a*.c       # a following by anything (including empty)  before .c
a.c      ax.c    axb.c    axy.c

$ ls a?.c       # a followed by exactly one character before .c
ax.c

$ ls ax?.c
axb.c    axy.c

$ ls ax*.c
ax.c     axb.c    axy.c

$ ls ax[abcf].c
axb.c
```

ls a??.c

ls ax??.c

Let's Do it

YORK UNIVERSITÉ UNIVERSITY

134

Used for filename wildcard, in ls, cp, mv, rm, cat, more, chmod ….
grep, find operating on multiple files

```
$ ls EECS2031*        # list files whose name beginning with EECS2031
EECS2031A             EECS2031B             EECS2031B.LAB03
EECS2031A.LAB01       EECS2031B.LAB01
EECS2031A.LAB02       EECS2031B.LAB02

$ ls EECS2031?    # files whose name is EECS2031 by exactly one char
EECS2031A     EECS2031B

$ ls EECS2031A*
EECS2031A     EECS2031A.LAB01    EECS2031A.LAB02

$ ls EECS2031A?
ls: No match.

$ ls EECS2031A.*
EECS2031A.LAB01    EECS2031A.LAB02

$ ls EECS2031?.*
EECS2031A.LAB01    EECS2031B.LAB01    EECS2031B.LAB03
EECS2031A.LAB02    EECS2031B.LAB02

$ ls EECS2031?.LAB?2
EECS2031A.LAB02    EECS2031B.LAB02
```

Same for other commands

135

find examples revisit

- $ find / -name x.c # search for file x.c in the entire file system
- $ find . -name '*.bak' # "*.bak" search for all bak files in current and subdirectories
- $ find . -name 'a?.c' # "a?.c" search for all aX.c

  a1.c
  a2.c
  a3.c # abc.c does not match

  Do a*c a??.c
- $ find . -name '*.c' -exec mv {} {}.2031 \;

  # find all c files and then rename it to filename.2031

  mv a1.c a1.c.2031
  mv lab3a.c lab3a.c.2031
  …..

136

grep Regex. Only place this course

grep a*b readme.txt more file12* cp file12* ./

	Regular expression	Filename substitution (wildcard)
a*	0 or more a	a followed by 0 or more anything
a?	0 or 1 a	a followed by exactly 1 anything
a+	1 or more a	
[abc] [a-c]	a or b or c	a or b or c

$ grep a*b file12*.c

Regex. 0 or more 'a' followed by 'b'
Match
b ab aab aaab aaaab
….

Wildcard. C file whose name begins with 'file12'
Match
file12.c file12A.c
file12AD.c file12ABEF.c
….

$ grep a?b file12?.c

Regex. 0 or 1 'a' followed by 'b'
Match b ab

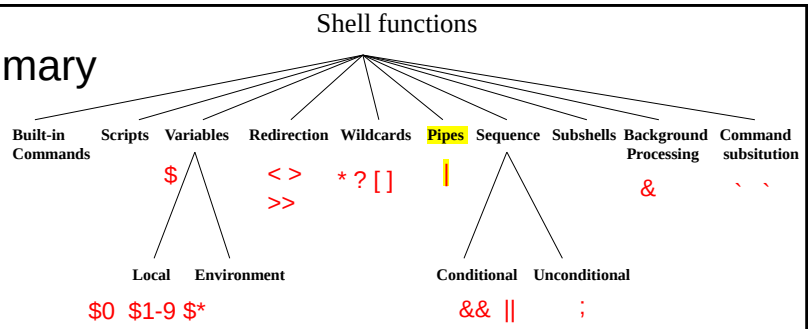
Wildcard. Match
file12A.c

138

YORK UNIVERSITÉ UNIVERSITY

138

Summary

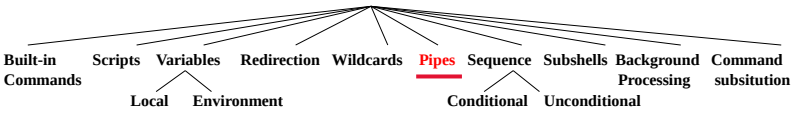


- Covered core shell functionality
  - Built-in commands/utilities
  - Redirection < > >>
  - Wildcards (filename substitution) * ? [ ]
  - Pipes |
  - Command substitution ` `
  - Sequence ; conditional sequence && ||
  - Background processing & Grouping ()
  - Variables $ variable substitution
  - Quoting ' ' " "
  - Scripts

YORK UNIVERSITÉ UNIVERSITY

139

Shell functions

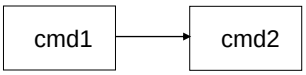


- **PIPES**

- Shells allow you to use the standard output of one process as the standard input of another process by connecting the processes together using the pipe(|) metacharacter.

- The sequence

$ command1 | command2



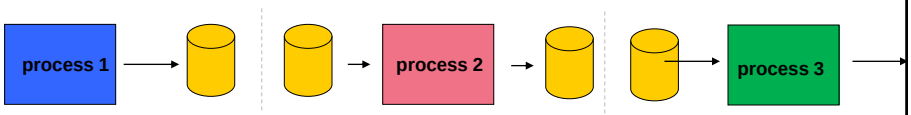
causes the standard output of command1 to "flow through" to the standard input of command2.

- Any number of commands may be connected by pipes.

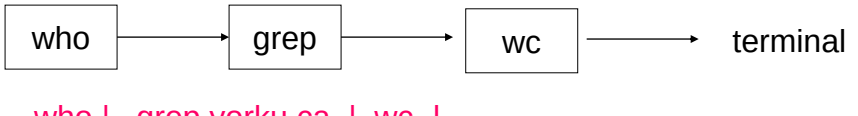
```
$ ls -l  | grep webapp        # who submitted using web submission?
$ who  | grep rogers          # who logon using rogers?
$ who  | grep bell  | wc -l   # how many using bell
```


Pipe-Equivalent Communication Using a File

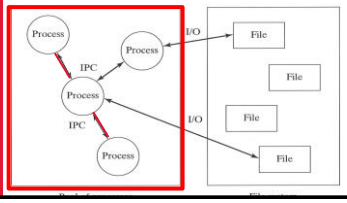
How many current users using yorku.ca network?

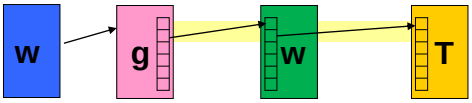


who > tmp; grep yorku.ca tmp > tmp2; wc -l tmp2

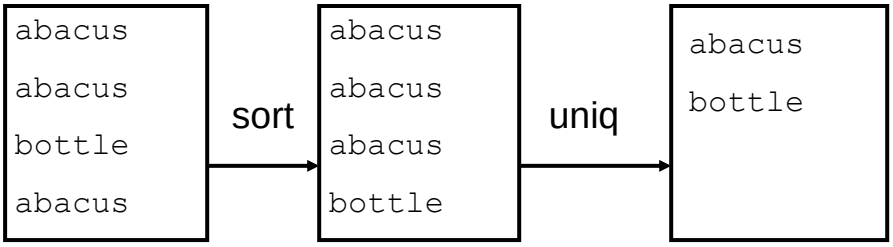


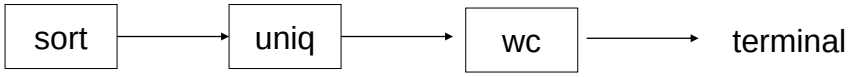
who | grep yorku.ca | wc -l



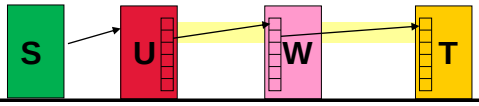


Pipeline Example



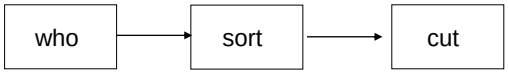


sort xFile123 | uniq | wc -l



143

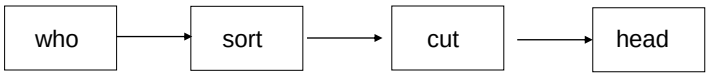
```
 $ who
giancarlo pts/1     2020-03-30 15:54  (cpef81d0f810383 …. cable.rogers.com)
andy      pts/2     2020-03-27 00:38  (cpeb8a386550d2d…..t.cable.rogers.com)
asalimi   pts/4     2020-03-29 19:51  (siren.eecs.yorku.ca)
kevinj22  pts/5     2020-03-27 18:58  (198-91-177-241.cpe.distributel.net)
……

$ who  |  sort –k 3 | cut  -d" " -f 1          # based on logon date
feshaghi
tmd12
burton
ulya
hina
navid
andy
mcnamee
omidvar
pmodheji
kevinj22
kevinj22
datta
….
 $ cat /etc/passwd  | grep –w Wang | wc -l
```



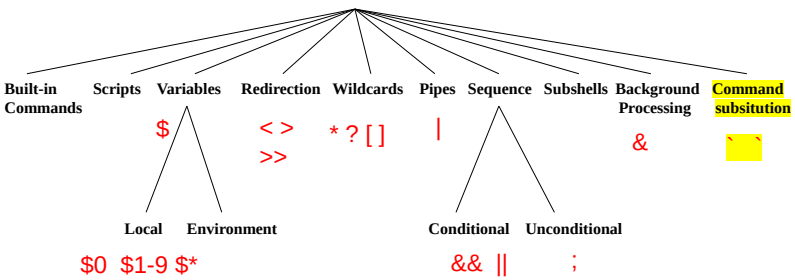
First 5 people logged on?

144

```
$ who
giancarlo pts/1     2020-03-30 15:54  (cpef81d0f810383 .... cable.rogers.com)
andy      pts/2     2020-03-27 00:38  (cpeb8a386550d2d.....t.cable.rogers.com)
asalimi   pts/4     2020-03-29 19:51  (siren.eecs.yorku.ca)
kevinj22  pts/5     2020-03-27 18:58  (198-91-177-241.cpe.distributel.net)
......

$ who  |  sort –k 3 | cut  -d" " -f 1  | head -5
feshaghi
tmd12
burton
ulya
hina

$
```



Shell functions



- Covered core shell functionality
  - Built-in commands/utilities
  - Redirection < > >>
  - Wildcards (filename substitution) * ? [ ]
  - Pipes |
  - Command substitution ` `
  - Sequence ; conditional sequence && ||
  - Background processing & Grouping ()
  - Variables $ variable substitution
  - Quoting ' ' " "
  - Scripts

Shell functions

Built-in Commands | Scripts | Variables (Local, Environment) | Redirection | Wildcards | Pipes | Sequence (Conditional, Unconditional) | Subshells | Background Processing | Command substitution

COMMAND SUBSTITUTION used very very … heavily in script!

A command surrounded by grave accents (`) - back quote - is executed, and its standard output is inserted in the command's place in the entire command line. Any new lines in the output are replaced by spaces.

`command`

For example:

```
$ echo  the date today is `date` , right?
the date today is Sun Mar 28 08:57:44 EDT 2020, right?
$
$ echo  there are `who | wc -l` users on the system
there are 31 users on the system
```

Bash: **$(command)**

```
echo  the date today is $(date), right?
```

148

Shell functions

Built-in Commands | Scripts | Variables (Local, Envirnment) | Redirection | Wildcards | Pipes | Sequence (Conditional, Unconditional) | Subshells | Background Processing | Command substitution

COMMAND SUBSTITUTION used very very … heavily in script!

A command surrounded by grave accents (`) - back quote - is executed, and its standard output is inserted in the command's place in the entire command line. Any new lines in the output are replaced by spaces.

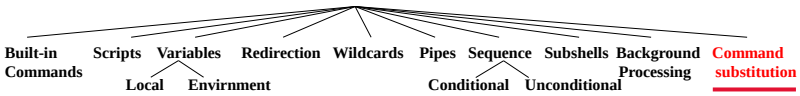
For example:

```
$ echo  there are `cat classlist | wc –l` students in the class
there are 153 students in the class

$ echo  has `cat classlist | grep –w Wang | wc -l` students name Wang
has 3 students name Wang
```

Bash: echo there are $(cat classlist | wc -l) students in the class

149

Shell functions



COMMAND SUBSTITUTION used very very … heavily in script!

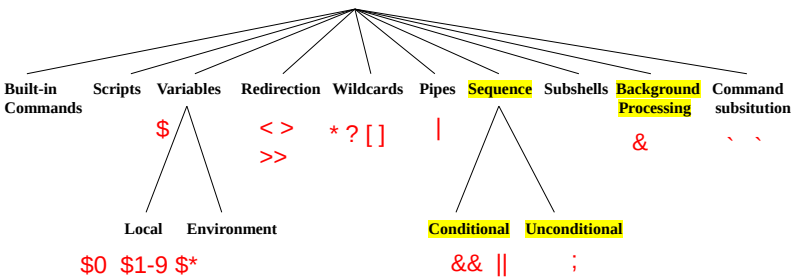
A command surrounded by grave accents (`) - back quote - is executed, and its standard output is inserted in the command's place in the entire command line. Any new lines in the output are replaced by spaces.

Two more examples:

```
$ which mkdir     # man which:  show the full pathname of shell command
/bin/mkdir
$ file `which mkdir`            # file /bin/mkdir
/bin/mkdir: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.32,
BuildID[sha1]=8cec890564feb596de5a36b1a5321b05a089079f, stripped

$ x=`date`    #  x=date ?

$ x=`cat classlist | wc –l `          # x get value  153 (talk later)
$ x=$(cat classlist | wc –l)        # x get value  153
```

150

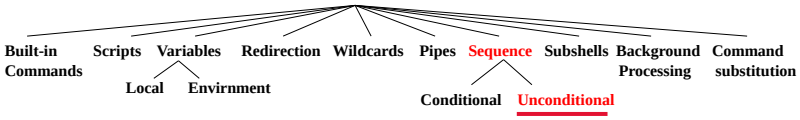
Shell functions



- Covered core shell functionality
  - Built-in commands/utilities
  - Redirection < > >>
  - Wildcards (filename substitution) * ? [ ]
  - Pipes |
  - Command substitution ` `
  - Sequence ; conditional sequence && ||
  - Background processing & Grouping ()
  - Variables $ variable substitution
  - Quoting ' ' " "
  - Scripts

151

Shell functions



- **SEQUENCES ; (unconditional)**

If you enter a series of simple commands or pipelines separated by semicolons, the shell will execute them in sequence, from left to right.

Here's an example:

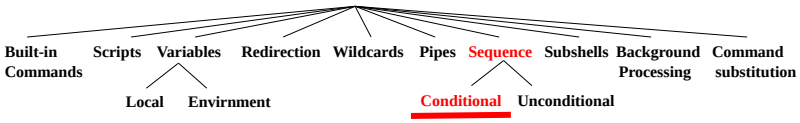
```
$ date;  pwd;  ls      # execute three commands in sequence.
Mon Feb  2  00:11:10  EDT  2019
/home/glass/wild
a.c    b.c    cc.c      dir1      dir2
$ _
$  gcc yourCode;   a.out > output.txt ;    cmp  output.txt  sampleSlu.txt
```

Each command in a sequence may be individually I/O redirected

```
$ date > date.txt;   pwd  > pwd.txt
```

152

Shell functions



- **Conditional Sequences** && ||

- Every UNIX process terminates with an exit value.
  By convention, an exit value of 0 means that the process completed successfully, and a > 0 exit value indicates failure.
  (opposite to C)

- You may construct sequences that make use of this exit value:

1) If you specify a series of commands separated by && tokens,
   cmd 1 && cmd2

cmd2 is executed only if cmd1 returns exit code of **0** i.e. cmd1 succeeded

2) If you specify a series of commands separated by || tokens,
   cmd 1 || cmd2

"Lazy evaluation"

153

cmd2 is executed only if cmd1 returns **nonzero** exit code. i.e., cmd1 fails

153

- For example,
  if gcc compiles a program without fatal errors,
  it creates an executable program called a.out and returns an exit code of 0;
  otherwise, it returns a non-zero exit code.

```
$ gcc  myprog.c  &&  a.out     # (only) if gcc successful, then run a.out
                               # otherwise, no run a.out

$ gcc  myprog.c  ||  echo  "compilation  failed."
                               # if gcc is not successful, then echo
```

```
$ grep –w Wang classlist  && echo "found someone in class"
$ grep –w WangXXX classlist  || echo "not found in class"
```

return 0 if match, return 1 otherwise

154

For your information

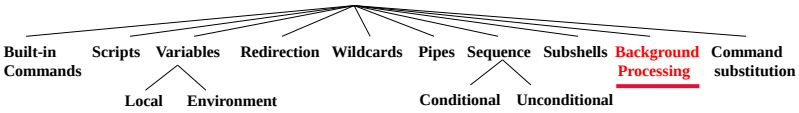
- **GROUPING COMMANDS ( )**
  - Commands may be grouped by placing them between parentheses, which causes them to be executed by a child shell(subshell).
  - The group of commands shares the same standard input, standard output, and standard error channels and may be redirected and piped as if it were a simple command.

- Here are some examples:

```
$ date;  ls;  pwd > out.txt          #  execute a sequence.
Sun Jul 21 23:25:26 EDT 2019        #  output from date.
 a.c          b.c                   #  output from ls.

$ cat  out.txt                      #  only pwd was redirected.
 /home/huiwang

$ ( date; ls; pwd ) > out.txt       # group and then redirect.

$ cat out.txt                       # all output was redirected.
Sun Jul 21 23:25:26 EDT 2019
a.c          b.c
/home/huiwang
```

155

Shell functions



- **Background Processing** &

- If you follow a simple command, pipeline, sequence of pipelines, or group of commands by the & metacharacter, a subshell is created to execute the commands as a background process
  $ jedit &

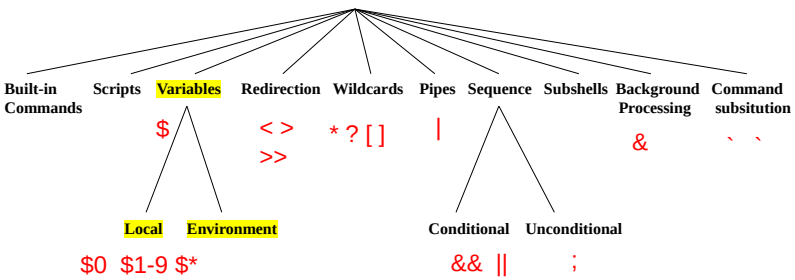
- The background process runs concurrently with the parent shell and does not take control of the keyboard.

- Background processing is therefore very useful for performing several tasks simultaneously, as long as the background tasks do not require input from the keyboard.

For your information

156

Shell functions



- Covered core shell functionality
  - Built-in commands/utilities
  - Redirection < > >>
  - Wildcards (filename substitution) * ? [ ]
  - Pipes |
  - Command substitution ` `
  - Sequence ; conditional sequence && ||
  - Background processing & Grouping ()
  - Variables $ variable substitution
  - Quoting ' ' " "
  - Scripts

today

157