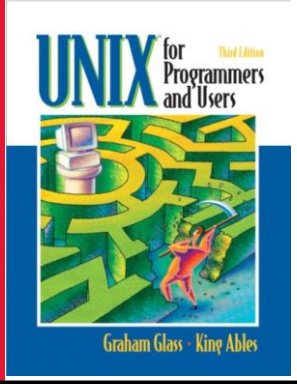
EECS2031 Software Tools

SU 2019

Jul 22, 2019 Lecture 10

1

UNIX



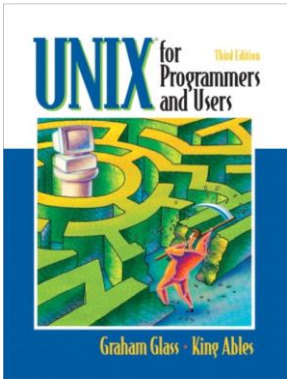
YORK
UNIVERSITÉ
UNIVERSITY

2

Contents

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

Previous lecture

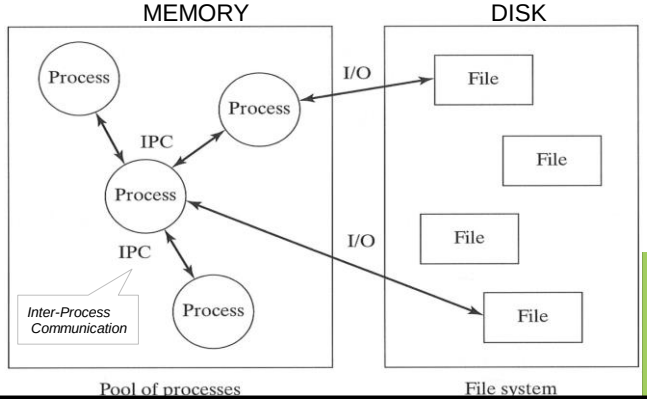


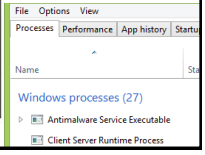
- Utilities/commands
  - Basic: pwd, ls, rmdir, mkdir, cat, more, mv, cp, rm, file, wc, chmod
  - Advanced: grep/egrep, sort, find ….
- Shell and shell scripting language

YORK UNIVERSITÉ UNIVERSITY

Files and Processes

- A file is a collection of data that is usually stored on DISK
- When a program is invoked, it is loaded from DISK into MEMORY. When a program is running (in MEMORY), it is called a process.
- Most processes read and write data from files.





File Pathnames

- Two files in different directories can have the same name. We need pathnames to differentiate between files with the same names located in different directories.

- A pathname is a sequence of directory names that leads you through the hierarchy from a starting directory to a target file.

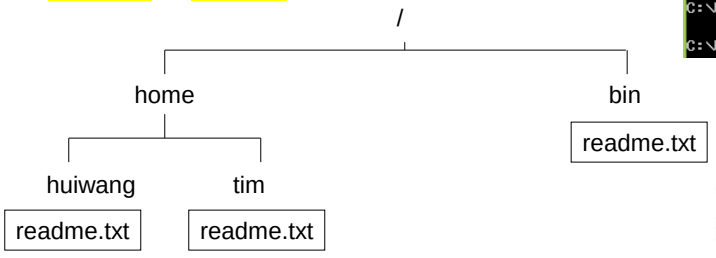
  gcc /cs/dept/course/2018-19/W/2031Z/submit/lab3/cse12345/lab3A.c .

  cat /home/tim/readme.txt

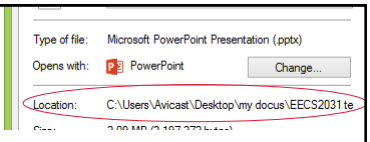
- Absolute or Relative

Windows uses \

```
C:\Users\Avicast>cd try

C:\Users\Avicast\try>dir
```



File: Absolute Pathnames

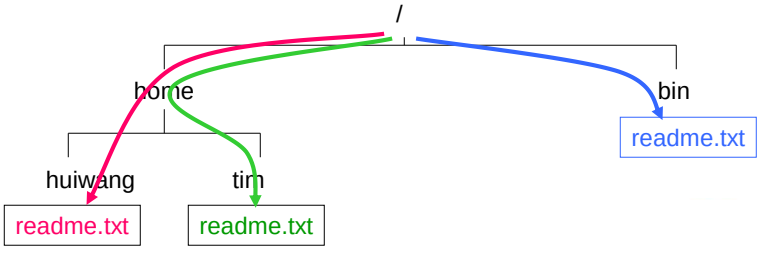


- A pathname starts from the root directory of file system is often termed an absolute, or full pathname.
- Valid from anywhere.

/home/huiwang/readme.txt ~/readme.txt

/home/tim/readme.txt

/bin/readme.txt



From anywhere, `cat /home/tim/readme.txt`

File: Relative Pathnames

- A process may also unambiguously specify a file by using a pathname relative to its current working directory.
- UNIX file system supports the following special fields that may be used when supplying a relative pathname:

Field	Meaning
.	current directory
..	parent directory

Same in DOS

```
cat ./input.txt    cat input.txt
./a.out  <  ../input.txt
rm  ../a1.c
cd ..
```

7

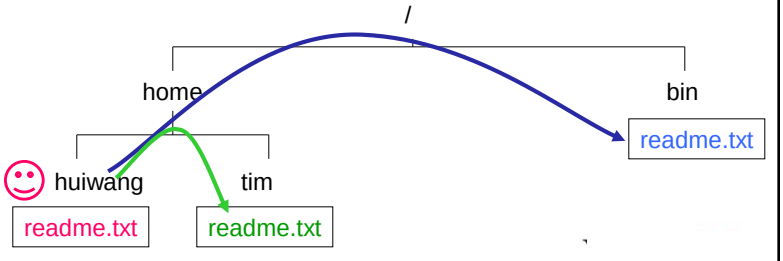
YORK UNIVERSITÉ UNIVERSITY

7

Relative Pathnames

- Relative Pathnames (from /home/huiwang)

cat readme.txt or ./readme.txt
cat ../tim/readme.txt
cat ../../bin/readme.txt



8

choose wisely cd ../../../../../../../../lab1 ???

8

File Permissions (Security)

- File permissions are the basis for file security. They are given in three clusters.

1 - rw- r-x r-- 1 huiwang faculty 213 Jan 31 00:12 heart.final

User (owner)	Group	Others
r w -	r - x	r - -

clusters

Each cluster of three letters has the same format:

Read permission	Write permission	Execute permission
r	w	x

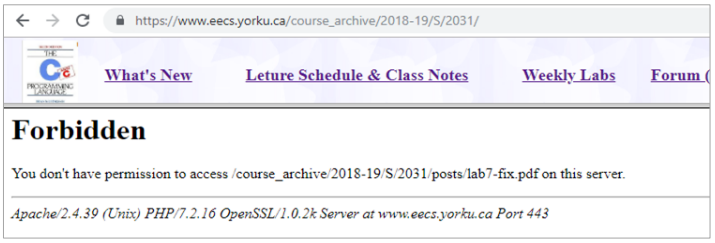
e.g., webfile: others need to have r permission
submit dir: group need to have w permission

9

9

Permission examples

webfile: others must has r permission -rwxr-x---



submit directory: group must has w permission -rwxr-xr--

```
sh-4.2$ submit 2031 lab6 lab7D0.c
error:  files may not be submitted for course 2031, assignment
lab6
Please contact your professor.
sh-4.2$
```

How to set/change permission?

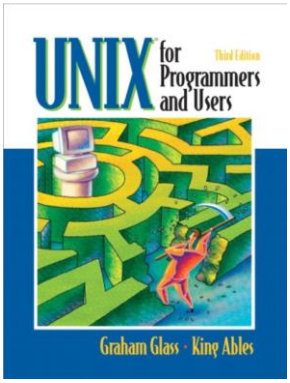
10

10

Contents

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

Previous lecture



- Utilities/commands
  - Basic: pwd, ls, rmdir, mkdir, cat, more, mv, cp, file, wc, chmod
  - Advanced: grep/egrep, sort, find ….
- Shell and shell scripting language

YORK UNIVERSITÉ UNIVERSITY

Processes

- Each command/utility involves a process
  - `ls, cd, pwd, gedit …`
  - Unix can execute many processes simultaneously.
- When a process ends, there is a return value aka exit code associated with the process outcome
  - a non-negative integer. ≥ 0

    0 means success

    anything > 0 represents various kinds of failure

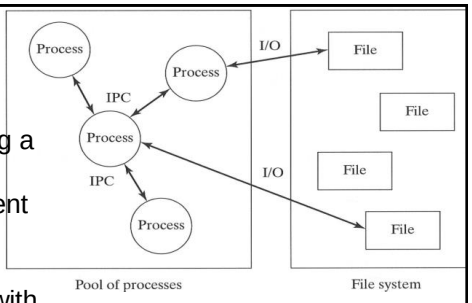
    Opposite to C
  - The return value is passed to the parent process

    Stored in system variable **$?**

```
sh-4.2$ pwd
/cs/home/huiwang
sh-4.2$ echo $?
0
sh-4.2$ date
Sat Mar 30 09:15:52 EDT 2019
sh-4.2$ echo $?
0
sh-4.2$
```

```
sh-4.2$ cd xxx
sh: cd: xxx: No such file or directory
sh-4.2$ echo $?
1
sh-4.2$ ls xxx
ls: cannot access xxx: No such file or directory
sh-4.2$ echo $?
2
sh-4.2$
```

Process: Communication



- Processes can communicate using a number of means:
  - passing arguments, environment
  - read/write regular disk files
  - exit values $?
  - inter-process communication with shared queues, memory and semaphores
  - signals
  - pipes
  - sockets

- A pipe is a one-way medium-speed data channel that allows two processes on the **same** machine to talk.

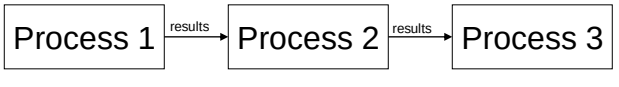
- If the processes are on different machines connected by a network, then a mechanism called a "socket" may be used instead. A socket is a two-way high-speed data channel.

Process: Unix Pipes

- A special mechanism called a "pipe" built into the heart of UNIX to support cascading utilities.

- A pipe allows a user to specify that the output of one process is to be used as the input to another process.

- Two or more processes may be connected in this fashion, resulting in a "pipeline" of data flowing from the first process through to the last.



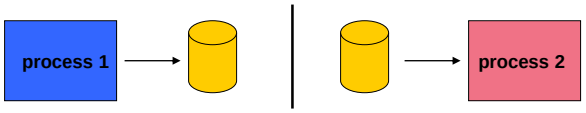
Pipeline Example

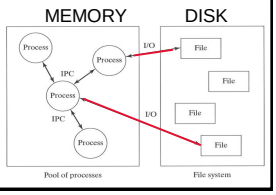
- A utility called **who** that outputs an unsorted list of the users, and another utility called **sort** that outputs a sorted version of its input.

A sorted list of users?

- Use a file (instead of a pipe)
  - Run first program, save output into disk file
  - Run second program, using file as input

```
$ who > tmp.txt
$ sort < tmp.txt
```



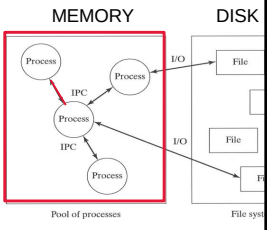


- Disadvantages:
  - Unnecessary use of the disk
    - Slow
    - Can take up a lot of space

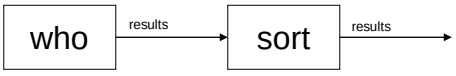
15

Pipeline Example

- A utility called **who** that outputs an unsorted list of the users, and another utility called **sort** that outputs a sorted version of its input.

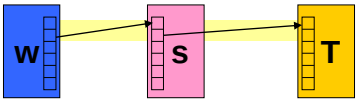


- Use a pipe instead of a file



- These two utilities may be connected together with a pipe so that the output from **who** passes directly into **sort**, resulting in a sorted list of users.

```
$ who | sort
```

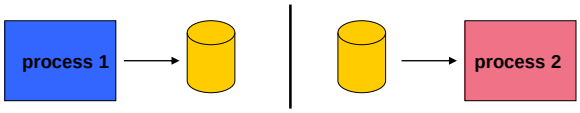


16

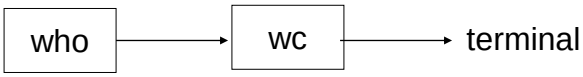
16

Pipe-Equivalent Communication Using a File

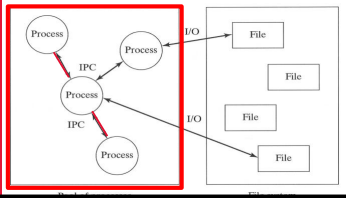
How many users are logged in?

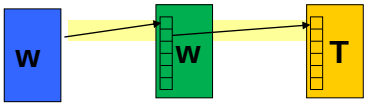


who > tmp.txt wc -l tmp.txt



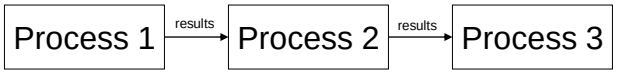
who | wc -l





17

More examples



- who | sort | head - 5 # list fist 5 people in the list
  who > tmp; sort tmp > tmp2; head -5 tmp2;
- wc -l EECS2031 # how many students
- cat EECS2031 | wc -l
- ls | more

dir /p or ***dir | more*** in DOS

18

YORK UNIVERSITÉ UNIVERSITY

18

Contents

- Overview of UNIX
  - Structures
  - File systems
    - absolute and relative pathname ../../input.txt
    - security -rwx--x--x
  - Process:
    - has return value 0 (success) or > 0 (sth wrong)
    - communication: pipes who | sort ls | more
- Utilities/commands
  - Basic
  - advanced
- Shell and shell scripting language

Basic utilities/commands

Introduced the following utilities, listed in in groups:

General	Directory	File	File print
man	pwd	cat	lp
clear	mkdir -p	more	lpr
echo	ls -d -S -t -r	head tail	lprm
date	cd	cp -r	lpq
	rmdir must be empty	mv move and/or rename	lpstat
		rm -r -i	
		file	
		wc -l -c -w	
		chmod g+w 750	
		chgrp	
		chown	
		newgrp	

```
red 302 % which rm
rm:      aliased to rm -i
red 303 %
```

```
red 303 % alias
cd      cd !* ; setXwd
cp      cp -i
l.      ls -d .* --color=auto
ll      ls -l --color=auto
ls      ls --color=auto
mc      source /usr/libexec/mc/mc-wrapper.csh
module  (set _prompt="$prompt";set prompt="";eval `/usr
/usr/bin/test 0 = $_exit;)
mv      mv -i
popd    popd ; setXwd
pushd   pushd !* ; setXwd
rm      rm -i
setXwd  /cs/local/bin/setXtermTitle "${HOST}:`pwd`"
vi      vim
red 304 %
```

In the login shell (tcsh), to be safe,

- When you issue cp, it is replaced by cp -i
- When you issue mv, it is replaced by mv -i
- When you issue rm, it is replaced by rm -i

In other shells, should use -i

cp vs mv

- Copy (copy+paste) and move (cut+paste) a 3G movie, which is faster?
- Below will both rename file1 to file2, what is the difference?

  cp file1 file2
  rm file1

  mv file1 file2

Counting Lines, Words and Chars in Files: WC

wc -lwc {fileName}*

- The wc utility counts the number of lines, words, and/or characters in a list of files.
- If no files are specified, standard input is used instead.

- -l option requests a line count,
- -w option requests a word count,
- -c option requests a character count.

- If no options are specified, then all three counts are displayed.

- A word is defined by a sequence of characters surrounded by tabs, spaces, or new lines.

Counting Lines, Words and Characters in Files: wc

- For example, to count lines, words and characters in the "heart.txt" file, we used:

```
$ wc  heart.txt        # obtain a count of the number of lines,
                       words, and characters.
  9     43     213    heart.txt
```

- Given class list file "EECS2031A", in which each line represents one student. How many students are there in the class? Let's do it

```
$ wc -l  EECS2031A
$ cat EECS2031A  |  wc  -l      # another way, using pipe
```

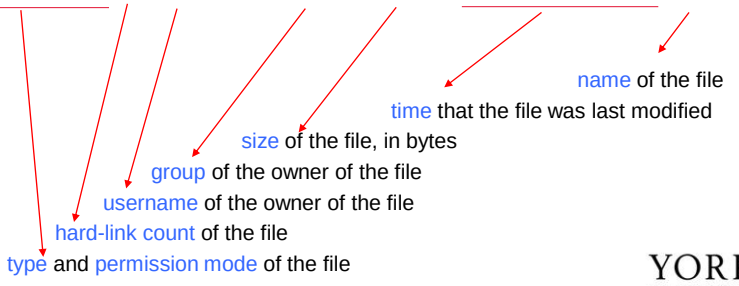
- How many people are currently logging onto eecs server?

```
$ who | wc -l          # using pipe
```

File Attributes

- We used ls to obtain a long listing of “heart.txt” and got the following output:

```
$ ls  -l       heart.txt
1  -rw-r--r--  1  huiwang faculty 213   Jan  31  00:12  heart.txt
$ ls  -ld      lyrics
1  drwxr-xr--  1  huiwang faculty 533   Jan  31  10:22  lyrics
$_
```




```
$  ls  -l       heart.txt
1  -rw-r--r--  1  huiwang faculty 213   Jan  31  00:12  heart.txt
```

File Attributes

Field #	Field value	Meaning
1	1	the number of blocks of physical storage occupied by the file
2	- rw-r--r--	the type and permission mode of the file, which indicates who can read, write, and execute the file
3	1	the hard-link count
4	huiwang	the username of the owner of the file
5	faculty	the group name of the file
6	213	the size of the file, in bytes
7	Jan 31 00:12	the time that the file was last modified
8	heart.final	the name of the file

File Attributes

- **File Types**
  - Field 2 describes the file's type and permission settings.

1 drwxr-xr-- 1 huiwang faculty 533 Jan 31 10:22 lyrics

1 - rw-r--r-- 1 huiwang faculty 213 Jan 31 00:12 heart.txt

- The first character of field 2 indicates the type of file, which is encoded as follows :

character	File Type
-	regular file
d	directory file
b	buffered special file(such as a disk drive)
c	unbuffered special file(such as a terminal)
l	symbolic link
p	pipe
s	socket

Determining Type of a File: file

file fileName(s)

- The file utility attempts to describe the contents of the fileName argument(s), including the language in which any of the text is written.

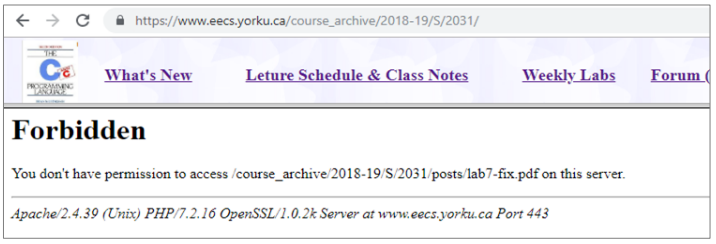
- not reliable; it may get confused.

```
$ file heart.txt          # determine the file type.
heart.txt: ASCII text
$ file lab5B.c
lab5B.c: C source, ASCII text
$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV) …..
```

Permission examples

Webfile accessible: others must has r permission

-rwxr-x---



submit directory open: group must has w permission

-rwxr-xr--

```
sh-4.2$ submit 2031 lab6 lab7D0.c
error:  files may not be submitted for course 2031, assignment
lab6
Please contact your professor.
sh-4.2$
```

29 How to set/change permission? chmod

Change File Permissions: chmod

Only owner and admin can change

chmod -R change{, change}* {fileName }+

- The **chmod** utility changes the **modes (permissions)** of the specified files according to the change parameters, which may take the following forms:

clusterSelection + newPermissions (add permissons)
clusterSelection - newPermissions (subtract permissions)
clusterSelection = newPermissions (assign permissions absolutely)

where **clusterSelection** is any combination of:

- **u** (user/owner)
- **g** (group)
- **o** (others)
- **a** (all)

rw- r-x r--
u g o
a

newPermissions is any combination of
r (read) **w (write)** **x (execute)**

30 The **-R** option recursively changes the modes of the files in directories.

rw- r-x r--
u g o
a

Changing File Permissions: examples

Requirement	Change parameters
Add group write permission	**g+w**
Remove group write permission	**g-w**
Remove other's read and write permission	**o-rw o-wr**
Add execute permission for user, group, and others.	**a+x u+x,g+x,o+x ugo+x**
Give the group read permission only.	**g=r**
Add write permission for user, and remove group read permission.	**u+w,g-r**
Give the other read and execute permission	**o=wx o=xw**

No space

For a web file to be accessible, o must has r permission.
$ chmod o+r lab7.pdf

31

Changing Permissions Using Numbers

- The chmod utility allows you to specify the new permission setting of a file as 3 octal numbers (0~7) .
- Each octal digit (0~7) represents a permission triplet.
  binary 1/0 1/0 1/0
  r w x

For example, if you wanted a file to have the permission settings of
rwx r-x --- # owner:rwx, group r x → chmod u=rwx, g=rx
then the octal permission setting would be **750**, calculated as follows:

	User	Group	Others
setting	rwx	r-x	---
binary	111	101	000
octal	7	5	0

32

32

Changing File Permissions Using Octal Numbers

- The octal permission setting would be supplied to chmod as follows:

```
$ chmod  750  lab4.pdf          # or  chmod u=rwx, g=rx  lab4.pdf
$ ls  -l  lab4.pdf              # confirm.
1 - rwx r-x ---      45   huiwang faculty    4096    Apr  29 14:35 lab4.pdf
$ _
```

7	5	0
111	101	000
rwx	r - x	- - -

Changing Permissions Using Octal Numbers

- The chmod utility allows you to specify the new permission setting of a file as an octal number.

rwx	7	Read, write and execute	111
rw-	6	Read, write	110
r-x	5	Read, and execute	101
r--	4	Read,	100
-wx	3	Write and execute	011
-w-	2	Write	010
--x	1	Execute	001
---	0	no permissions	000

chmod u=rwx,g=rwx,o=rx	chmod 775
chmod u=rwx,g=rx,o=	chmod 750
chmod u=rw,g=r,o=r	chmod 644
chmod u=rw,g=r,o=	chmod 640
chmod u=rw,go=	chmod 600
chmod u=rwx,go=	chmod 700

An example: setting up submit directory using chmod

- https://wiki.eecs.yorku.ca/dept/tdb/services:submit:submit-setup

Department of Electrical Engineering & Computer Science

Technical Database

- News
- Departmental Services
- E-Mail
- Lab Schedules
- Login and Remote Access
- Operating System
- Policies and Procedures
- Printing
- Scanning
- Software
- Web Publishing
- Wiki Publishing

SUBMIT DIRECTORY SETUP

Technical Database » Departmental Services » Submit » Submit Directory Setup

In order to setup a submit directory for your course:

- The course directory must be under /eecs/course.
- In the course directory, create a directory called "submit". That should be accessible by everyone.
- Under the submit folder, create one directory per assignment. The assignment directory must be writable by group, not by "other".

For example, to setup a submit directory for course 1021 and assignment a1, use the following c

```
% mkdir /eecs/course/1021 <- this is only necessary if you haven't creat
% chmod 755 /eecs/course/1021
% mkdir /eecs/course/1021/submit
% chmod 755 /eecs/course/1021/submit
% mkdir /eecs/course/1021/submit/a1
% chgrp submit /eecs/course/1020/submit/a1
% chmod 770 /eecs/course/1021/submit/a1
```

or chmod u=rwx,g=rwx a1
or chmod ug=rwx a1

rwxrwx---

If you no longer wish to allow submissions for an assignment (e.g. past a due date) then remove directory:

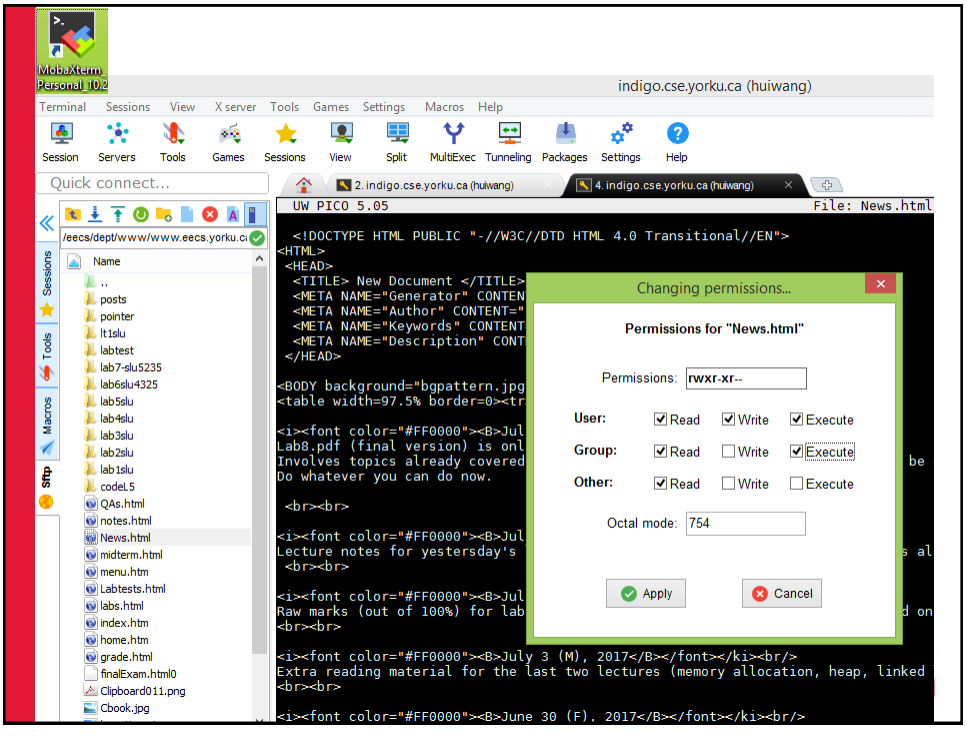
```
chmod g-w /eecs/course/1021/submit/a1
```

or chmod 750 a1

rwxr-x---

35

35



36

Basic utilities/commands

Introduced the following utilities, listed in in groups:

General	Directory	File	File print
man	pwd	cat	lp
clear	mkdir -p	more	lpr
echo	ls -d -S -t -r	head tail	lprm
date	cd	cp -r	lpq
	rmdir must be empty	mv move and/or rename	lpstat
		rm -r -i	
		file	
		wc -l -c -w	
		chmod g+w 750	
		chgrp	
		chown	
		newgrp	

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
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, cut, 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

Filtering Files grep, uniq

- **grep, egrep**, **fgrep** "Global/Get Regular Expression and Print"

  -w -i -v

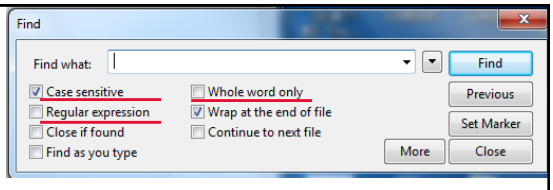
  - Filter out, all lines that do not contain a specified pattern,
  - Giving you the line that contains the specified pattern

```
$ cat inputFile.txt                # list the file to be filtered
line1 Well you know it's your bedtime,
line2 So turn off the light,
line3 Say all your prayers and then,
line4 Oh you sleepy young heads dream of wonderful things,
line5 Beautiful mermaids will swim through the sea,
line6 And you will be swimming there too.

$ grep  the  inputFile.txt         # search for the word "the"
line2 So turn off the light,
line3 Say all your prayers and then,
line5 Beautiful mermaids will swim through the sea,
line6 And you will be swimming there too.
```

39

39

Searching for Regex: grep

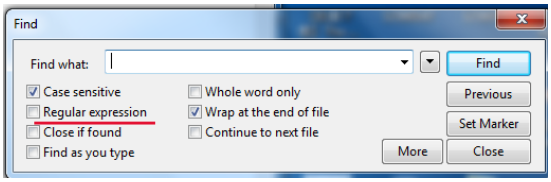



```
$ grep -w  the inputFile.txt       #  -w: Whole word  only
line2 So turn off the light,
line5 Beautiful mermaids will swim through the sea,

$ grep -v  -w  the inputFile.txt   #  -v:  reverse the filter.
line1 Well you know it's your bedtime,
line3 Say all your prayers and then,
line4 Oh you sleepy young heads dream of wonderful things,
line6 And you will be swimming there too.

$ grep -i -w the inputFile.txt     # ignore case,  default case sensitive

$ grep -w Wang EECS2031A           # who have family name Wang?
$ grep -w Wang EECS2031A  |  wc -l          # how many ?
```

40

40

Searching for Regex: grep



How to use grep to search lines that contain numbers?

$ grep ? inputFile.txt

How to use grep to search lines that contain lower case letters?

$ grep ? inputFile.txt

Given String s = "abs0deb2afg43affe6wqf53sd5", how to replace all digits with character 'X' in Java

41

41

Utility	Kind of pattern that may be searched for
fgrep	fixed string only
grep	regular expression
egrep	extended regular expression

Regular Expressions

YORK UNIVERSITÉ UNIVERSITY

42

What is a Regular Expression?

- A regular expression (regex) describes a pattern to match multiple input strings.

- Regular expressions are endemic to Unix
  - Some utilities/programs that use Regex:
    - **vi**, **ed**, **sed**, and **emacs**
    - **awk**, **tcl**, **perl** and **Python**
    - **grep**, **egrep**
    - **Compilers** **scanf (" %[^\n]s ", str);**

  For this course

- The simplest regular expression is a string of literal characters to match.

- The string **matches** the regular expression if it contains the substring.

Regular Expressions: Exact Matches

regular expression → cks $ grep cks inputFile.txt

UNIX Tools rocks.

match

UNIX Tools sucks.

match

UNIX Tools is okay.

no match

Regular Expressions: Multiple Matches

- A regular expression can match a string in more than one place.

regular expression → apple $ grep apple inputFile.txt

Scrapple from the apple.

match 1 *match 2*

$ grep -w apple inputFile.txt ?

Regular Expressions: Matching Any Character

- The . regular expression can be used to match any character.

regular expression → o. $ grep o. inputFile.txt

Force me to put on that

match 1 *match 2*

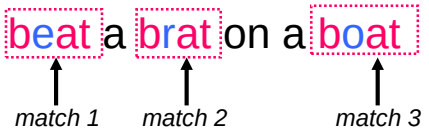
$ grep -w o. inputFile.txt ?

Regular Expressions: Alternate Character Classes

- Character classes `[]` can be used to match any specific set of characters.

regular expression → b [eor] a t

$ grep b[eor]at inputFile.txt



- [aeiou] will match any of the characters a, e, i, o, u
- [kK]orn will match korn or Korn

47

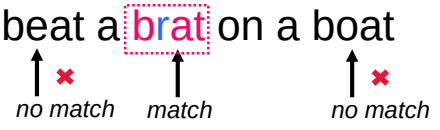
47

Regular Expressions: Negated Character Classes

- Character classes can be negated with the `[^]` syntax.

regular expression → b [^eo] a t

$ grep b[^eo]at inputFile.txt



scanf (" %[^\n]s", str);

48

48

Regular Expressions: Other Character Classes

- Other examples of character classes:
  - [0123456789] will match any digit
  - [abcde] will match a b c d e

- Ranges can also be specified in character classes

  [0-9] is the same as [0123456789]
  [a-e] is equivalent to [abcde]

  $ grep [0-9] inputFile.txt

- You can also combine multiple ranges

  [abcde123456789] is equivalent to [a-e1-9]
  [a-zA-Z] all the letters

49

Regular Expressions: Named Character Classes

- Commonly used character classes can be referred to by name
  - alpha,
  - lower,
  - upper,
  - alnum,
  - digit,
  - punct,
  - cntl

For your information

- Syntax [[:name:]]
  - [0-9] [[:digit:]] $ grep [[:digit:]] inputFile
  - [a-zA-Z] [[:alpha:]]
  - [a-zA-Z0-9] [[:alnum:]]
  - [45a-z] [45[:lower:]]

- Important for portability across languages

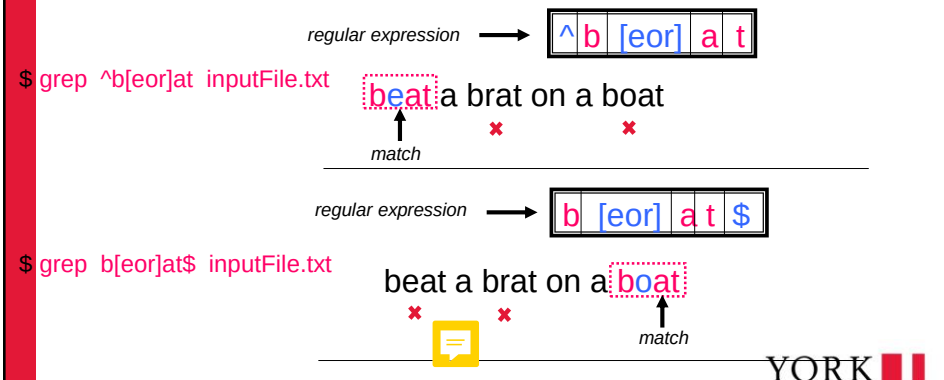
50

YORK UNIVERSITÉ UNIVERSITY

50

Regular Expressions: Anchors

- Anchors are used to match at the beginning or end of a line (or both).
  - ^ means beginning of the line ^ the begin with the
  - $ means end of the line the$ end with the



51

Regular Expressions: Anchors

- Anchors are used to match at the beginning or end of a line (or both).
  - ^ means beginning of the line ^ the begin with the
  - $ means end of the line the$ end with the

$grep cse EECS2031A

```
indigo 339 % grep cse EECS2031A
cse****          *********        Yu   Ying
cse******        *********        Lee  JunXu
eqao             cse******        Tong Treacy
indigo 340 %
```

$grep ^cse EECS2031A

```
indigo 340 % grep ^cse EECS2031A
cse****          *********        Yu   Ying
cse******        *********        Lee  JunXu
indigo 341 %
```

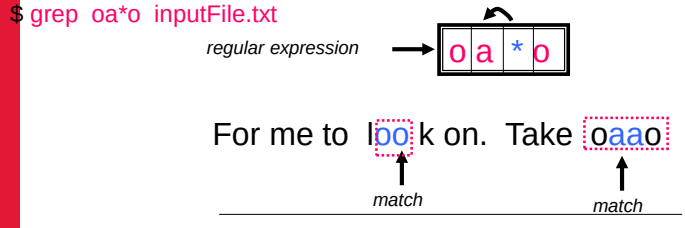
52

Regular Expression: Repetitions

"Kleene Star"

- The * is used to define **zero or more** occurrences of the *single* regular expression preceding it.

zero or more occurrences of 'a' (between y z)
yz yaz yaaz yaaaz



zero or more occurrences of 'a' (between o o)
oo oao oaao oaaao

Regular Expressions: Repetition Ranges, Subexpressions

- Ranges can also be specified
  - `{n,m}` notation can specify a range of repetitions for the immediately preceding regex
  - `{n}` means exactly *n* occurrences
  - `{n,}` means at least *n* occurrences
  - `{n,m}` means at least *n* occurrences but no more than *m* occurrences

- Example:

For your information

```
.{0,}  same as .*
a{2,}  same as aaa*     # at least 2 occurrences
a{2}   same as aa       # exact 2 occurrences
```

- If you want to group part of an expression so that * applies to more than just the previous character, use ( ) notation

- Subexpresssions are treated like a single character

  `a*` matches zero or more occurrences of `a`

  `abc*` matches `ab`, `abc`, `abcc`, `abccc`, … # ab followed by 0 or more c

  `(abc)*` matches `abc`, `abcabc`, `abcabcabc`, …

Regular Expressions: Repetition Ranges, Subexpressions

- Some examples

Regular Expression	Matches
"a*"	ZERO or more 'a'
"ba*"	b, ba, baa, baaa, baaaa, ...
"a*b*"	Ø, a, aaa, aaab, abbb, b, bbb, ... zero or more 'a', followed by zero or more 'b'

~~abbab~~

- Don't get confused with filename wildcard *
  - ls a*.c a followed by 0 or more any char -- anything
  - ls ba* ba followed by 0 or more any char -- anything

55

Extended Regular Expressions: Repetition Shorthands

- The * (star) has already been seen to specify zero or more occurrences of the immediately preceding character

- The ? (question mark) specifies an optional character, the single character that immediately precedes it
  - July? will match Jul or July zero or one occurrence
    - Equivalent to (Jul|July)
  - abc?d will match abd and abcd
    but will not match abccd ✘

- The + (plus) means one or more occurrence of the preceding character
  - abc+d will match abcd, abccd, or abccccccd
    but will not match abd ✘

56

Repetition recap

	Regular expression
a*	0 or more a
a?	0 or one a
a+	1 or more a

- ab*c matches ac abc abbc abbbc abbbbc ….
- ab?c matches ac abc
- ab+c matches abc abbc abbbc abbbbc ….

YORK UNIVERSITÉ UNIVERSITY

57

grep and egrep RE

Pattern	Maning	Example	
c	Non-special, matches itself	'tom'	
\c	Turn off special meaning	'\$'	
^	Start of line	'^ab'	anchored
$	End of line	'ab$'	anchored
.	Any single character	'.nodes'	
[…]	Any single character in []	'[tT]he'	
[^…]	Any single character not in []	'[^tT]he'	
R*	Zero or more occurrences of R	'e*'	repetition
R?	Zero or one occurrences of R (egrep)	'e?'	repetition
R+	One or more occurrences of R (egrep)	'e+'	repetition
R1R2	R1 followed by R2	'[st][fe]'	
R1|R2	R1 or R2 (egrep)	'the|The'	

58

58

grep and egrep RE

Pattern	Maning	Example
c	Non-special, matches itself	'tom'
\c	Turn off special meaning	'\$'
^	Start of line	'^ab'
$	End of line	'ab$'
.	Any single character	'.nodes'
[…]	Any single character in []	'[tT]he'
[^…]	Any single character not in []	'[^tT]he'
R*	Zero or more occurrences of R	'e*'
R?	Zero or one occurrences of R (egrep)	'e?'
R+	One or more occurrences of R (egrep)	'e+'
R1R2	R1 followed by R2	'[st][fe]'
R1|R2	R1 or R2 (egrep)	'the|The'

Don't get confused

anchored

repetition

59

59

grep and egrep RE

Pattern	Maning	Example
c	Non-special, matches itself	'tom'
\c	Turn off special meaning	'\$'
^	Start of line	'^ab'
$	End of line	'ab$'
.	Any single character	'.nodes'
[…]	Any single character in []	'[tT]he'
[^…]	Any single character not in []	'[^tT]he'
R*	Zero or more occurrences of R	'e*'
R?	Zero or one occurrences of R (egrep)	'e?'
R+	One or more occurrences of R (egrep)	'e+'
R1R2	R1 followed by R2	'[st][fe]'
R1|R2	R1 or R2 (egrep)	'the|The'

Don't get confused

anchored

repetition

Don't get confused with UNIX metacharacter (file name wildcards)

ls file*.c *.java
cp xFile?.c . one any

60

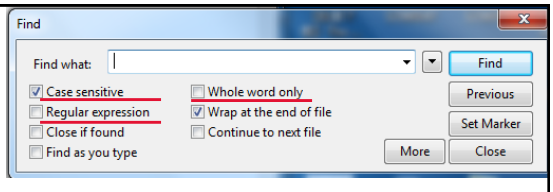
60

Utility	Kind of pattern that may be searched for
fgrep	fixed string only
grep	regular expression
egrep	extended regular expression

- Regular expression and extended expression maybe confusing.
- grep may behave differently in different shells.

- So for this course
  - Use grep -E or egrep
  - Work on Bourne (again) shell (sh/bash)

Searching for Regex: grep



$ grep ^[tT]he inputFile.txt # begins with the or The

$ grep [0-9]x inputFile.txt # contains digits followed by 'x'

$ grep ^[a-z] inputFile.txt # begins with a lower case letter

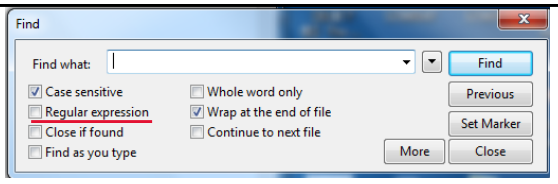
$ grep .nd inputFile.txt # contains one any character followed by nd

$ grep [ab]nd$ inputFile.txt # ends with 'and' or 'bnd'

$ grep -w W[ao]ng EECS2031 ?

$ grep -w W[ao]ng EECS2031 | wc -l # how many ?

Searching for Regex: grep



How to use grep to search lines that contain numbers?

$ grep [0-9] inputFile.txt # or grep [[:digit:]] inputFile.txt

How to use grep to search lines that contain lower case letters?

$ grep [a-z] inputFile.txt # or grep [[:lower:]] inputFile.txt

Given String s= "abs0deb2afg43affe6wqf53sd5", how to replace all digits with character 'X' in Java?

```
s = s.replaceAll("[0-9]", "X");
```

replaceAll

```
public String replaceAll(String regex,
                         String replacement)
```

Replaces each substring of this string that matches the given regular expression with the given replacement.

63

63

Exit code of grep/egrep

Matching found: 0 No matching: 1 No such file 2

```
$ grep Wang EECS2031
$ echo $?          # display its exit value.
0                  # indicates success.

$ grep Leung  EECS2031
$ echo $?
1                  # indicates failure (not matching).

$grep Wang  classlistX
grep: classlistX: No such file or directory
$ echo $?
2                  # indicates failure (not such a file).
```

Look for man
man grep | grep –w “exit”

Used in scripting

YORK UNIVERSITÉ UNIVERSITY

64

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

65

65

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
```

66

How about merging un-adjacent lines too? sort and then uniq

66

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)

67

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

68

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

```
$ sort -k2   data.txt          #  -k 2        sort by column 2, surname
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 -k3   data.txt          #  -k 3        sort by field/column 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
```

Lexicographically column 3 not sorted correctly

```
$ sort -k3 -n  data.txt        #  -nk3  -nk 3
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 -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

71

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

$ sort  -k5 -M data.txt          #  +4 -5
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

$ sort  -k5 -M -r data.txt       #  +4 -5
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
```

Lexicographically
Months not sorted
correctly

-M enables
months to be sorted
correctly

-r reverse
descending

72

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)
```


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

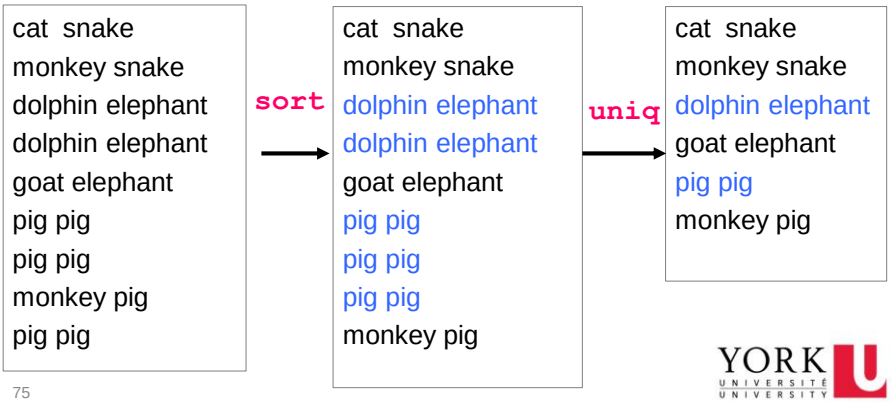
```
halt:x:7:0:halt:/sbin:/sbin/halt
operator:x:11:0:operator:/root:/sbin/nologin
root:x:0:0:root:/root:/bin/bash
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
sync:x:5:0:sync:/sbin:/bin/sync
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin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



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
- The cmp utility determines whether two files are the same.

```
$ 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
$ _
```

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.

```
$ 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

77

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

78

```
$ 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 1-3  data.txt > data2.txt
```

79

find Utility

find pathList expression

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

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

- 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 all the Java class files and delete them"

```
find . -name "*.class"  -exec rm {} \;
```

80

find Utility

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

81

find examples

- $ find / -name x.c # search for file/dir named x.c in the entire file syster
- $ find . -mtime 14 # search for files/dir modified in the last 14 days in current and subdirectories
- $ find . -name '*.bak' # "*.bak" search for all bak files in current and subdirectories
- $ find . -name 'a?.c' # "a?.c" search for all file/dir named aX.c
  a1.c
  a2.c
  a3.c
- $ find . -name 'a?.c' | wc -l # how many

- $ find . **-type f -maxdepth 1** -name 'lab*' # files only, name starts 'lab
  -type d # directory only in current directory only (no subdirectories)

82

find examples -exec

- $ find . -name '*.bak' -exec rm {} \;
  # remove all files that end with .bak

- $ find . -name 'a?.c' -exec mv {} {}.bak \;
  # find aX.c files and then rename them to aX.c.bak

- $ find . -name '*.c' -exec cp {} {}.2019SU \;
  # find all c files and then copy it to filename.c.2019SU

- $ find . -name '*.c' -exec chmod 770 {} \;
  # find all c files and change mode to rwxrwx---

YORK UNIVERSITÉ UNIVERSITY

83

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
Extract 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

YORK UNIVERSITÉ UNIVERSITY

84

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

85

Contents

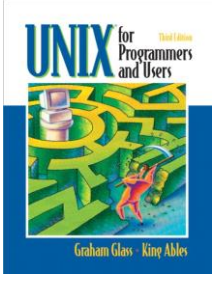
- Overview of UNIX
  - Structures
  - File systems
    - absolute and relative pathname
    - security -rwx--x--x
  - Process:
    - has return value 0 (success) or non 0 (sth wrong)
    - communication: pipes who | sort who | grep Wang | wc -l
- Utilities/commands
  - Basic, mkdir, cat, more, cp, rm, mv, file, wc, chmod
  - Advanced grep/egrep, uniq, sort, diff/cmp, cut, find,
- Shell (common shell functionalities)
- Bourn (again) Shell
  - scripting language

86

YORK UNIVERSITÉ UNIVERSITY

86

UNIX Shells



Ch 4 Unix shells
"UNIX for Programmers and Users"
Third Edition, Prentice-Hall, GRAHAM GLASS, KING ABLES

87

- **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)

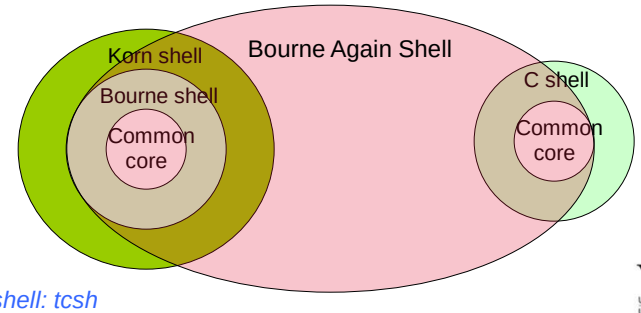
YORK UNIVERSITÉ UNIVERSITY

88

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:

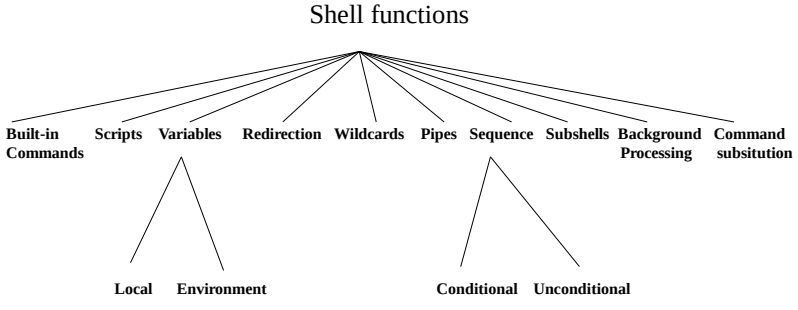


Login shell: tcsh
An enhanced but based on and completely compatible version of the C shell, *csh*

89

SHELL FUNCTIONALITY

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



90

- **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

YORK UNIVERSITÉ UNIVERSITY

91

- **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.
*	File-substitution (wildcard); matches zero or more characters.
?	File-substitution (wildcard); matches any single character.
[…]	File-substitution (wildcard); matches any character between the brackets.

CSE1020 lab tour. Don't confuse with RE

92

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.
<<tok	Input redirection; reads standard input from script up to tok.
' ' " "	quoting

93

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

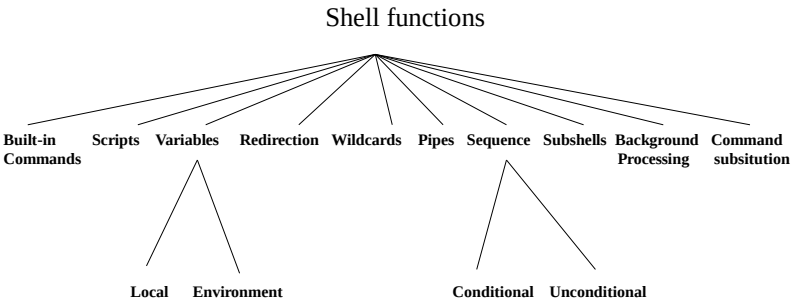
  When all metacharacters have been processed, the command is finally executed.

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

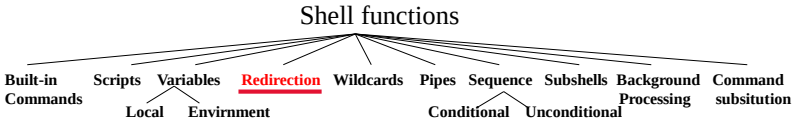
- Here's an example:

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

$ echo hi \> file2     #   inhibit > metacharacter.
hi > file2             #    > is treated like other characters.
$ cat file2            #    look at the file again.  Not written
ls: cannot access file2: No such file or directory such a file
```

94



Now, lets go through the functionalities, as well as their associated metacharacters



- **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  EECS2031 > names.txt

$ echo "new line"   >   fileName
$ echo "new line"   >> fileName
```

Difference?

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

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.

YORK UNIVERSITÉ UNIVERSITY

97

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.	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 file123.* | grep a?.c file123?

98

Used for filename wildcard, in ls, cp, mv, rm, cat, more, chmod …

Here are some examples of wildcards in action:

```
$ ls   *.c       # list any text ending in ".c "
a.c      b.c      ax.c

$ ls   ?.c       # list text for which one character is 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/2018-19/S/2031/xFile?   .

$ cp /eecs/dept/course/2018-19/S/2031/xFile*   .

$ cp /eecs/dept/course/2018-19/S/2031/xFile[23]  .
```

99

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
- $ find . -name '*.c' -exec cp {} {}.2019SU \;

  # find all c files and then copy it to filename.2019SU

100

grep RE. Only place this course

	Regular expression	Filename substitution (wildcard)
a*	0 or more a	a followed by 0 or more anything
a?	0 or one a	a followed by 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
```

RE. 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
....

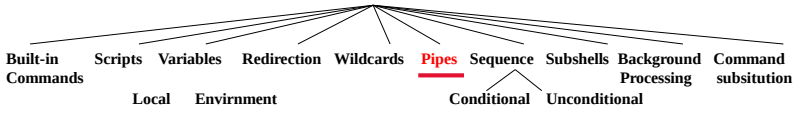
```
$ grep  a?b    file12?.c
```

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

Wildcard. Match
file12A.c

YORK UNIVERSITÉ UNIVERSITY

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
```

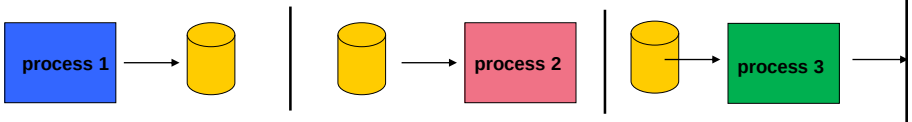
causes the standard output of command1 to "flow through" to the standard input of command2.

- Any number of commands may be connected by pipes.

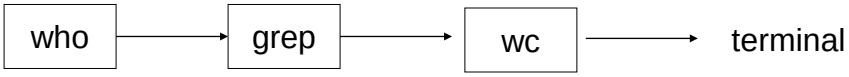
YORK UNIVERSITÉ UNIVERSITY

Pipe-Equivalent Communication Using a File

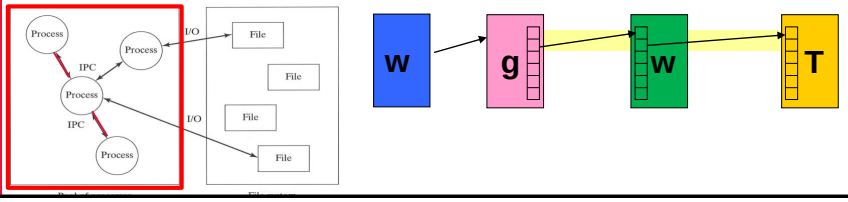
How many users have name Wang?



who > tmp; grep Wang tmp > tmp2; wc -l tmp2

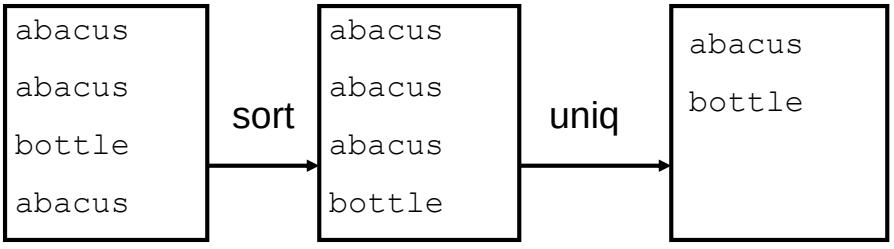


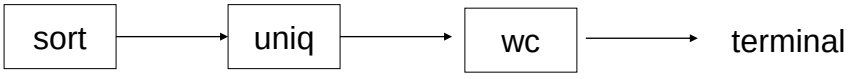
who | grep Wang | wc -l



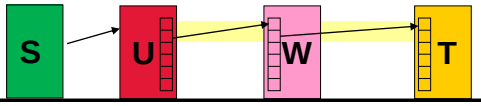
103

Pipeline Example



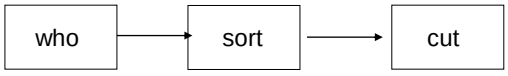


sort xFile123 | uniq | wc -l



104

```
 $ who
giancarlo pts/1     2019-06-30 15:54  (cpef81d0f810383 …. cable.rogers.com)
andy      pts/2     2019-06-27 00:38  (cpeb8a386550d2d…..t.cable.rogers.com)
asalimi   pts/4     2019-06-29 19:51  (siren.eecs.yorku.ca)
kevinj22  pts/5     2019-06-27 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
….
```

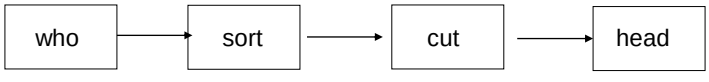


First 5 people logged on?

YORK UNIVERSITÉ UNIVERSITY

105

```
 $ who
giancarlo pts/1     2019-06-30 15:54  (cpef81d0f810383 …. cable.rogers.com)
andy      pts/2     2019-06-27 00:38  (cpeb8a386550d2d…..t.cable.rogers.com)
asalimi   pts/4     2019-06-29 19:51  (siren.eecs.yorku.ca)
kevinj22  pts/5     2019-06-27 18:58  (198-91-177-241.cpe.distributel.net)
……

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

$
```



YORK UNIVERSITÉ UNIVERSITY

106

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 the date today is `date` , right?
the date today is Sun Jul 20 08:57:44 EDT 2019, right?
$ _
$ echo there are `who | wc -l` users on the system
there are 31 users on the system

YORK UNIVERSITÉ UNIVERSITY

107

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:

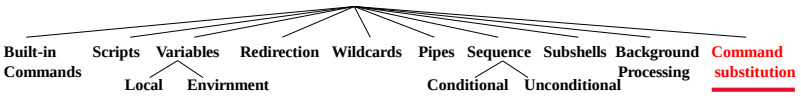
$echo there are `cat EECS2031 | wc –l` students in the class
there are 135 students in the class

$echo has `cat EECS2031 | grep –w Wang | wc -l` students name Wang
has 4 students with name Wang

YORK UNIVERSITÉ UNIVERSITY

108

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:

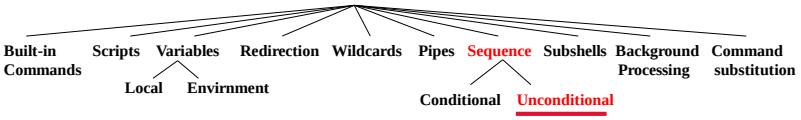
```
$ which mkdir     # man which:  show the full pathname of shell command
/bin/mkdir
$ file `which mkdir`            # file /bin/mkdir
/bin/mkdir: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses
shared libs), for GNU/Linux 2.6.32,
BuildID[sha1]=8cec890564feb596de5a36b1a5321b05a089079f,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

$x=`wc –l classlist`          # x get value  135 (talk later)
```

For your information

YORK UNIVERSITÉ UNIVERSITY

109

Shell functions



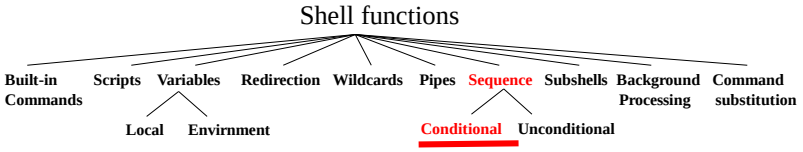
- **SEQUENCES ;**

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
```

110



- **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 an exit code of **0**. i.e., successful

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

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

111

- 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    # if gcc successful, then run a.out

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

$ grep –w Wang classlist    &&   echo "found someone in the list"
```

return 0 if match, return 1 otherwise

YORK UNIVERSITÉ UNIVERSITY

112

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).

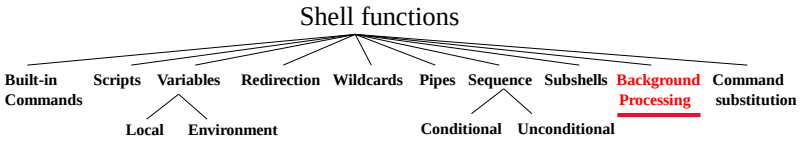
- The group of commands shares the same standard input, standard output, and standard error channels and may be redirected and piped as if it were a simple command.

- 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
```

YORK UNIVERSITÉ UNIVERSITY

113



- **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 &
```

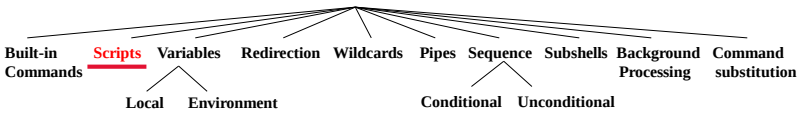
- The background process runs concurrently with the parent shell and does not take control of the keyboard.

- 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

YORK UNIVERSITÉ UNIVERSITY

114



SHELL PROGRAMS: SCRIPTS

Any series of shell commands may be stored inside a regular text file for later execution.

A file that contains shell commands is called *a script*.

- batch file （.bat） in Windows

Before you can run a script, you must give it execute permission

chmod u+x filename

To run it, you need only to type its name.

```
echo hello world
date
```

Scripts are useful for storing commonly used sequences of commands, and they range in complexity from simple one-liners to fully blown programs.

115

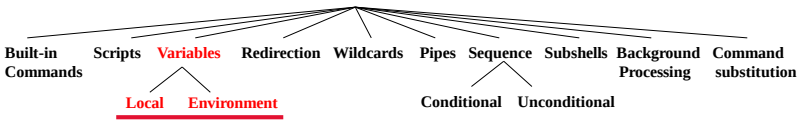
- **SHELL PROGRAMS: SCRIPTS**

```
$ cat  >  script.sh                    # create  the bash script.
#! /bin/sh
# This is a sample sh script.
echo "Hello world"
echo  The date today is `date`.
^D                                     #  end of input.

$ chmod u+x  script.sh          # make the scripts executable.

$ script.sh          # execute the shell script.
hello world
The date today is Sun Jul 21  19:50:00  EDT  2019
```

` ` command substitution

YORK UNIVERSITÉ UNIVERSITY

116



- **VARIABLES and variable substitution $**

- A shell supports two kinds of variables:
  local and environment variables.

  local:
  user defined,
  positional

Both kinds of variables hold data in a string format.

The child shell gets a copy of its parent shell's environment variables, but not its local variables.

Every shell has a set of predefined environment variables that are usually initialized by the startup files.

117

For your information

- **Environment VARIABLES (for your reference)**

- Here is a list of the predefined environment variables that are common to all shells:

Name	Meaning
$HOME	the full pathname of your home directory
$PATH	a list of directories to search for commands
$MAIL	the full pathname of your mailbox
$USER	your username
$SHELL	the full pathname of your login shell
$TERM	the type of your terminal

118

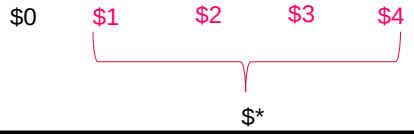
Built-in local variables

For your information

-several common built-in local variables that have special meanings:

Name	Meaning
$$	The process ID of the shell.
$0	The name of the shell script (if applicable).
$1..$9	$n refers to the n'th command line argument (if applicable).
$*	A list of all the command-line arguments.

$ myscript paul ringo george john



YORK UNIVERSITÉ UNIVERSITY

119

variable substitution $

```
$ x=5
$ echo value of x is $x      # value of x is 5


$ name=Graham
$ echo Hi, I am $name    # Hi, I am Graham

$ echo $?
```

YORK UNIVERSITÉ UNIVERSITY

120

QUOTING

There are often times when you want to inhibit the shell's wildcard-substitution * ? [], variable-substitution $, and/or command-substitution ` ` mechanisms.

The shell's quoting system allows you to do just that.

- Here's the way that it works:

1) Single quotes (' ') inhibits both wildcard substitution, variable substitution, and command substitution.

2) Double quotes(" ") inhibits wildcard substitution only.

121

QUOTING

- The following example illustrates the difference between the two different kinds of quotes:

```
$ echo  3 * 4 = 12        # remember, * is a wildcard.
3   a.c    b    b.c    c.c    4 = 12
```

```
$ echo  "3 * 4 = 12"     # double quotes inhibit wildcards.
3 * 4 = 12

$ echo  ' 3 * 4 = 12 '     # single quotes inhibit wildcards.
3 * 4 = 12
```

another way?

```
$ echo  3 \* 4 = 12      # backslash inhibit a metacharacter
3 * 4 = 12
```

YORK UNIVERSITÉ UNIVERSITY

122

```
$ name=Graham     # assign value to name variable

$ echo  3 * 4 = 12, my name is $name - today is `date`
 3 a.c   b   b.c   c.c 4 = 12, my name is Graham - today is Sun Jul 21
```

123

```
$ name=Graham     # assign value to name variable

$ echo  3 * 4 = 12, my name is $name - today is `date`
 3 a.c   b   b.c   c.c 4 = 12, my name is Graham - today is Sun Jul 21
```

- By using **single quotes (apostrophes)** around the text,
  we inhibit all wildcarding and variable and command substitutions:

```
$ echo  '3 * 4 = 12, my name is $name - today is `date`'
```

?

124

```
$ name=Graham     # assign value to name variable

$ echo  3 * 4 = 12, my name is $name - today is `date`
 3 a.c  b  b.c  c.c 4 = 12, my name is Graham - today is Sun Jul 21
```

- By using **single quotes (apostrophes)** around the text, we inhibit all wildcarding and variable and command substitutions:

```
$ echo  '3 * 4 = 12, my name is $name - today is `date`'
 3 * 4 = 12, my name is $name - today is `date`
$ _
```

inhibited

125


```
$ name=Graham     # assign value to name variable

$ echo  3 * 4 = 12, my name is $name - today is `date`
 3 a.c  b  b.c  c.c 4 = 12, my name is Graham - today is Sun Jul 21
```

- By using **single quotes (apostrophes)** around the text, we inhibit all wildcarding and variable and command substitutions:

```
$ echo  '3 * 4 = 12, my name is $name - today is `date`'
 3 * 4 = 12, my name is $name - today is `date`
$ _
```

inhibited

- By using **double quotes around** the text, we inhibit wildcarding, but allow variable and command substitutions:

```
$ echo "3 * 4 = 12, my name is $name - today is `date`"
```

?

126

```
$ name=Graham     # assign value to name variable

$ echo  3 * 4 = 12, my name is $name - today is `date`
 3 a.c  b  b.c  c.c 4 = 12, my name is Graham - today is Sun Jul 21
```

- By using **single quotes (apostrophes)** around the text, we inhibit all wildcarding and variable and command substitutions:

```
$ echo  '3 * 4 = 12, my name is $name - today is `date`'
 3 * 4 = 12, my name is $name - today is `date`
$ _
```

inhibited

- By using **double quotes around** the text, we inhibit wildcarding, but allow variable and command substitutions:

```
$ echo "3 * 4 = 12, my name is $name - today is `date`"
3 * 4 = 12, my name is Graham - today is Sun Jul 21 23:25:26 EDT
$ -
```

inhibited

interpreted

YORK UNIVERSITÉ UNIVERSITY

127

- Here's the way that it works:

1) Single quotes (' ') inhibits wildcard substitution, variable substitution, and command substitution.

2) Double quotes(" ") inhibits wildcard substitution only.

```
$ x=5
$ echo "value of x is $x"
value of x is 5

$ echo  "there are `who | wc -l` people logged on"
there are 32 people logged on
```

" " does not inhibit variable substitution $
" " does not inhibit command substitution

Both ' ' and " " inhibit wildcard substitution * ?

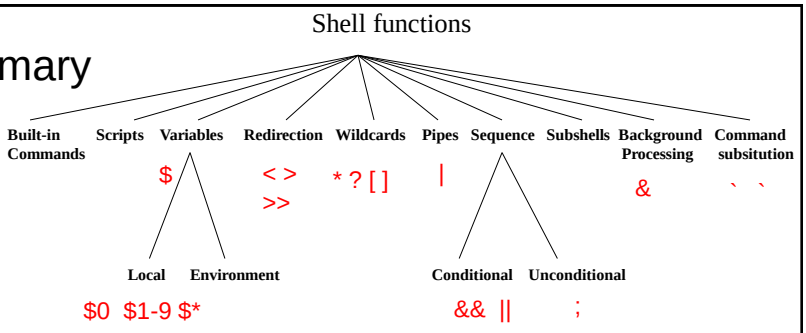
Needed for Some shell e.g., tcsh	$ egrep the lyrics	$ egrep 'the' lyrics	$ egrep "the" lyrics
	~~$ egrep ab? lyrics~~	$ egrep "ab?" lyrics	$ egrep 'ab?' lyrics
	~~$ egrep ab*c lyrics~~	$ egrep "ab*c" lyrics	$ egrep 'ab*c' lyrics
needed	$ find . -name lyrics	$ find . -name 'lyrics'	$ find . -name "lyrics"
	~~$ find . -name a?.c~~	$ find . -name 'a?.c'	$ find . -name "a?.c"
	~~$ find . -name *.c~~	$ find . -name '*.c'	$ find . -name "*.c"

128

Better to always " " wildcard substitution * ?

128

Summary



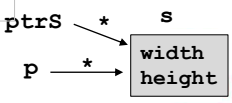
- Covered core shell functionality
  - Built-in commands/utilities
  - Redirection < > >>
  - Wildcards (filename substitution) * ? [ ]
  - Pipes |
  - Command substitution ` `
  - Sequence ; conditional sequence && ||
  - Background processing & Grouping ()
  - Variables $ variable substitution
  -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

Contents

- Overview of UNIX
  - Structures
  - File systems
    - absolute and relative pathname
    - security -rwx--x--x
  - Process:
    - has return value 0 (success) or non 0 (sth wrong)
    - communication: pipes who | sort who | grep Wang | wc -l
- Utilities/commands
  - Basic mkdir, cat, cp, rm, mv, file, wc, chmod
  - Advanced grep/egrep, uniq, sort, diff/cmp, cut, find,
- Shell (common shell functionalities)
- Bourn (again) Shell
  - scripting language

```
struct shape {
    float width;
    float height;
};
main(){
   struct shape r = {1,3};
   struct shape s = r;
   struct shape * ptrS = &s;
   do_sth (ptrS);
   printf("%d %d", r.width, r.height);
   printf("%d %d", s.width, s.height);
}

void do_sth(struct shape *p)
{
     p -> width  += 100;
     p -> height += 200;
}
```

```
1   3
101 203
```



131