CS 35L

Week 2

TA: Tomer Weiss
Jan-11-2016

PTE

- Will pass sign-in sheet
- Attend each class and sign your name
- PTE given during second week of classes

Seasnet

Secure Remote Login File Transfer

For secure remote login and
file transfer, use ssh and sftp (instead of telnet and ftp).

To run graphical
application on a remote unix server, see X11 Forwarding.

Windows Clients

- PuTTY SSH
  - How to install
  - How to use
- WinSCP freeware SFTP and SCP client for Windows
- X11 Forwarding
- Xming X Server for Windows

Unix Clients

- Example: how to use ssh
- Example: how to use sftp

Macintosh Clients

- Note that Mac OS X includes OpenSSH by default.
- OpenSSH Mac OS clients

www.seasnet.ucla.edu/secure-remote-login-file-transfer/

Seasnet important notice

- Login and do your Homework on the following servers:
  - ssh username@lnxsrv07.seas.ucla.edu
  - ssh username@lnxsrv09.seas.ucla.edu

- Seasnet is upgrading other seasnet servers

- We are going to test your assignment solutions on these servers

Seasnet and VPN

- Login and do your Homework on the following servers:
  - ssh username@lnxsrv07.seas.ucla.edu
  - ssh username@lnxsrv09.seas.ucla.edu

- lnxsrv07 doesn't require VPN

Assignment 2

- Deadline - 01-15-2016

- This week's assignment is going to be **consuming**

Assignment 2

- Deadline - 01-15-2016

- This week's assignment is going to be **very consuming**

- Start early -- it will help you a lot!

- Sample shell that can help:

    - https://ccle.ucla.edu/mod/forum/view.php?id=976217

Shell Scripting and Regular Expressions

Week 2

Regular Expressions

Lab setup - *Locale* for Assignment 2

- Please set your Locale:
  - <u>export</u> LC_ALL='C'

- Important because we want the 'sort' shell command to be ASCII character complainant
  - Otherwise your output for 'sort' is unknown and not deterministic, and your assignment results will not be as expected

Sorting words

- Investigate the 'sort' command
- man sort
- sort all the words in
  - /usr/share/dict/words
- save to your home folder
- sort [option] /usr/share/dict/words …?

The tr command (1)

- First, download assign2.html

  - **wget** http://web.cs.ucla.edu/classes/winter16/cs35L/assign/assign2.html

- **How to run: tr -c 'A-Za-z' '[\n*]'**
  ?

Basic I/O Redirection

- Most programs read from stdin
- Write to stdout
- Send error messages to stderr

```
$ cat                    With no arguments, read
   standard input, write              standard output
now is the time          Typed by the user
now is the time          Echoed back by cat
for all good men
for all good men
to come to the aid of their country
to come to the aid of their country
^D                       Ctrl-D, End of file
```

Redirection and Pipelines

- Use *program < file* to make *program*'s standard input be *file*:

  ```
  cat < file.txt
  ```

- Use *program > file* to make *program*'s standard output be *file*:

  ```
  cat < file.txt > file2.txt
  ```

- Use *program >> file* to send *program*'s standard output to the end of *file*.

- Use *program1 | program2* to make the standard output of *program1* become the standard input of *program2*.

  ```
  cat assign2.html | tr -c 'A-Za-z' '[\n*]'
  ```

The tr command (2)

- First, download assign2.html

  - **wget** http://web.cs.ucla.edu/classes/winter16/cs35L/assign/assign2.html

- Question: What does tr do?

The tr command (3)

- First, download assign2.html

  - `wget` http://web.cs.ucla.edu/classes/winter16/cs35L/assign/assign2.html

- Question: What does tr do?
  – Filters everything except characters from A to Z **and** from a to z
  – 'A-Za-z' is a regular expression

Searching for Text

- grep: Uses basic regular expressions (BRE)
- egrep: Extended grep that uses extended regular expressions (ERE)
  - grep -E
  - egrep
  - sed -r
- Fgrep: Fast grep that matches fixed strings instead of regular expressions.
  - grep -F
  - fgrep

Simple grep

```
$ who                         Who is logged on
tolstoy tty1 Feb 26 10:53
tolstoy pts/0 Feb 29 10:59
tolstoy pts/1 Feb 29 10:59
tolstoy pts/2 Feb 29 11:00
tolstoy pts/3 Feb 29 11:00
tolstoy pts/4 Feb 29 11:00
austen pts/5 Feb 29 15:39 (mansfield-park.example.com)
austen pts/6 Feb 29 15:39 (mansfield-park.example.com)

$ who | grep -F austen        Where is austen logged on?
austen pts/5 Feb 29 15:39 (mansfield-park.example.com)
austen pts/6 Feb 29 15:39 (mansfield-park.example.com)
```

Regular Expressions

- Notation that lets you search for text that fits a particular criterion, such as "starts with the letter a"

- http://regexpal.com to test your regex expressions

- Simple regex tutorial:
  https://www.icewarp.com/support/online_help/203030104.htm

Regular expressions

Character	BRE / ERE	Meaning in a pattern
\	Both	Usually, turn off the special meaning of the following character. Occasionally, enable a special meaning for the following character, such as for \(...\) and \{...\}.
.	Both	Match any single character except NUL. Individual programs may also disallow matching newline.
*	Both	Match any number (or none) of the single character that immediately precedes it. For EREs, the preceding character can instead be a regular expression. For example, since . (dot) means any character, .* means "match any number of any character." For BREs, * is not special if it's the first character of a regular expression.
^	Both	Match the following regular expression at the beginning of the line or string. BRE: special only at the beginning of a regular expression. ERE: special everywhere.

Regular Expressions (cont'd)

$	Both	Match the preceding regular expression at the end of the line or string. BRE: special only at the end of a regular expression. ERE: special everywhere.
[...]	Both	Termed a bracket expression, this matches any one of the enclosed characters. A hyphen (-) indicates a range of consecutive characters. (Caution: ranges are locale-sensitive, and thus not portable.) A circumflex (^) as the first character in the brackets reverses the sense: it matches any one character not in the list. A hyphen or close bracket (]) as the first character is treated as a member of the list. All other metacharacters are treated as members of the list (i.e., literally). Bracket expressions may contain collating symbols, equivalence classes, and character classes (described shortly).
\{*n,m*\}	BRE	Termed an *interval expression*, this matches a range of occurrences of the single character that immediately precedes it. \{*n*\} matches exactly n occurrences, \{*n*,\} matches at least n occurrences, and \{*n,m*\} matches any number of occurrences between n and m. n and m must be between 0 and RE_DUP_MAX (minimum value: 255), inclusive.
\(\)	BRE	Save the pattern enclosed between \(and \) in a special *holding space*. Up to nine subpatterns can be saved on a single pattern. The text matched by the subpatterns can be reused later in the same pattern, by the escape sequences \1 to \9. For example, **\(ab\).*\1** matches two occurrences of ab, with any number of characters in between.

Regular Expressions (cont'd)

n	BRE	Replay the nth subpattern enclosed in \\(and \\) into the pattern at this point. n is a number from 1 to 9, with 1 starting on the left.
{*n,m*}	ERE	Just like the BRE \\{*n,m*\\} earlier, but without the backslashes in front of the braces.
+	ERE	Match one or more instances of the preceding regular expression.
?	ERE	Match zero or one instances of the preceding regular expression.
|	ERE	Match the regular expression specified before or after.
( )	ERE	Apply a match to the enclosed group of regular expressions.

Examples

Expression	Matches
tolstoy	The seven letters tolstoy, anywhere on a line
^tolstoy	The seven letters tolstoy, at the beginning of a line
tolstoy$	The seven letters tolstoy, at the end of a line
^tolstoy$	A line containing exactly the seven letters tolstoy, and nothing else
[Tt]olstoy	Either the seven letters Tolstoy, or the seven letters tolstoy, anywhere on a line
tol.toy	The three letters tol, any character, and the three letters toy, anywhere on a line
tol.*toy	The three letters tol, any sequence of zero or more characters, and the three letters toy, anywhere on a line (e.g., toltoy, tolstoy, tolWHOtoy, and so on)

POSIX Bracket Expressions

Class	Matching characters	Class	Matching characters
[:alnum:]	Alphanumeric characters	[:lower:]	Lowercase characters
[:alpha:]	Alphabetic characters	[:print:]	Printable characters
[:blank:]	Space and tab characters	[:punct:]	Punctuation characters
[:cntrl:]	Control characters	[:space:]	Whitespace characters
[:digit:]	Numeric characters	[:upper:]	Uppercase characters
[:graph:]	Nonspace characters	[:xdigit:]	Hexadecimal digits

Backreferences

- Match whatever an earlier part of the regular expression matched
  - Enclose a subexpression with \(and \).
  - There may be up to 9 enclosed subexpressions and may be nested
  - Use \digit, where digit is a number between 1 and 9, in a later part of the same pattern.

Pattern	Matches
\(ab\)\(cd\)[def]*\2\1	abcdcdab, abcdeeecdab, abcdddeeffcdab, ...
\(why\).*\1	A line with two occurrences of why
\([[:alpha:]_][[:alnum:]_]*\) = \1;	Simple C/C++ assignment statement

Matching Multiple Characters with One Expression

*	Match zero or more of the preceding character
\\{*n*\\}	Exactly n occurrences of the preceding regular expression
\\{*n*,\\}	At least n occurrences of the preceding regular expression
\\{*n,m*\\}	Between n and m occurrences of the preceding regular expression

Anchoring text matches

Pattern	Text matched (in bold) / Reason match fails
ABC	Characters 4, 5, and 6, in the middle: abc**ABC**defDEF
^ABC	Match is restricted to beginning of string
def	Characters 7, 8, and 9, in the middle: abcABC**def**DEF
def$	Match is restricted to end of string
[[:upper:]]\{3\}	Characters 4, 5, and 6, in the middle: abc**ABC**defDEF
[[:upper:]]\{3\}$	Characters 10, 11, and 12, at the end: abcDEFdef**DEF**
^[[:alpha:]]\{3\}	Characters 1, 2, and 3, at the beginning: **abc**ABCdefDEF

Operator Precedence (High to Low)

Operator	Meaning
[. .] [= =] [: :]	Bracket symbols for character collation
\metacharacter	Escaped metacharacters
[ ]	Bracket expressions
\(\) \digit	Subexpressions and backreferences
* \{ \}	Repetition of the preceding single-character regular expression
no symbol	Concatenation
^ $	Anchors

sed

- Now you can extract, but what if you want to replace parts of text?
- Use sed!

`sed 's/`**`regExpr`**`/`**`replText`**`/'`

- Example

```
sed 's/:.*//' /etc/passwd  Remove everything
       after the first colon
```

Text Processing Tools

- sort: sorts text
- wc: outputs a one-line report of lines, words, and bytes
- lpr: sends files to print queue
- head: extract top of files
- tail: extracts bottom of files

More on Variables

- **Read only command**

```
hours_per_day=24 seconds_per_hour=3600 days_per_week=7      Assign values
readonly hours_per_day seconds_per_hour days_per_week   Make read-only
```

- **Export: puts variables into the environment, which is a list of name-value pairs that is available to every running program**

```
PATH=$PATH:/usr/local/bin        Update PATH

export PATH                      Export it
```

- **env: used to remove variables from a program's environment or temporarily change environment variable values**

- **unset: remove variable and functions from the current shell**

Parameter Expansion

- ## Process by which the shell provides the value of a variable for use in the program

```
reminder="Time to go to the dentist!" Save value in
              reminder
sleep 120                         Wait two minutes
echo $reminder                    Print message
```

Pattern–matching operators

path=/home/tolstoy/mem/long.file.name

Operator	Substitution
${*variable#pattern*}	If the pattern matches the beginning of the variable's value, delete the shortest part that matches and return the rest.
Example: ${path#/*/}	Result: tolstoy/mem/long.file.name
${*variable##pattern*}	If the pattern matches the beginning of the variable's value, delete the longest part that matches and return the rest.
Example: ${path##/*/}	Result: long.file.name
${*variable%pattern*}	If the pattern matches the end of the variable's value, delete the shortest part that matches and return the rest.
Example: ${path%.*}	Result: /home/tolstoy/mem/long.file
${*variable%%pattern*}	If the pattern matches the end of the variable's value, delete the longest part that matches and return the rest.
Example: ${path%%.*}	Result: /home/tolstoy/mem/long

String Manipulation

- ${string:position}: Extracts substring from $string at $position

- ${string:position:length} Extracts $length characters of substring $string at $position

- ${#string}: Returns the length of $string

The Shell and OS

The Shell and OS

- The shell is the user's interface to the OS
- From it you run programs.
- Common shells
  - bash, zsh, csh, sh, tcsh
- Allow more complex functionality then interacting with OS directly
  - Tab complete, easy redirection

Scripting Languages Versus Compiled Languages

- Compiled Languages
  - Ex: C/C++, Java
  - Programs are translated from their original source code into object code that is executed by hardware
  - Efficient
  - Work at low level, dealing with bytes, integers, floating points, etc

- Scripting languages
  - Interpreted
  - Interpreter reads program, translates it into internal form, and execute programs

Why Use a Shell Script?

- Simplicity

- Portability

- Ease of development

Example

```
$ who
george      pts/2    Dec 31 16:39 (valley-forge.example.com)
betsy       pts/3    Dec 27 11:07 (flags-r-us.example.com)
benjamin    dtlocal  Dec 27 17:55 (kites.example.com)
jhancock    pts/5    Dec 27 17:55 (:32)
Camus       pts/6    Dec 31 16:22
tolstoy     pts/14   Jan 2 06:42

$ who | wc -l        Count users
6


$ who | grep tomer   Where is tomer?
6
```

Idea

- Build a script that searches for a name
  - i.e. $who | grep userWeAreLookingFor

- Check if userWeAreLookingFor is logged in

- Let's create it!
  - create a file called `finduser`

finduser

Script:

```
#! /bin/sh
# finduser --- see named by first arguement is
  logged in
who | grep $1
```
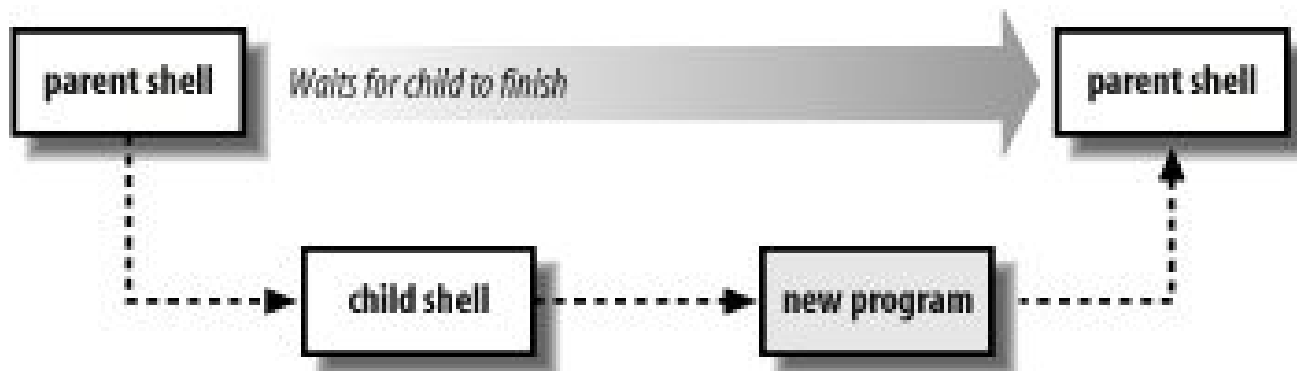
Run it:
$ **chmod +x finduser**   *Make it executable*

$ **./finduser tomer**

The #! First Line

- A shell script is just a file with shell commands.
- When the shell runs a program (e.g finduser), it asks the kernel to start a new "child process" and run the given program in that process.
- First line is used to state which "child shell" to use:

  #! /bin/csh –f
  #! /bin/awk –f
  #! /bin/sh

Ubuntu Shell Scripting

- Ubuntu 6.01+ uses by defualt "dash" shell which is POSIX complaint
- /bin/sh isa link to /bin/dash
- "dash" and "bash" should not have any differences in use
- Bash tutorial
  - http://linuxconfig.org/bash-scripting-tutorial

Variables

- Start with a letter or underscore and may contain any number of following letters, digits, or underscores

- Declared using =

  - Var='hello world'

- Referenced with $

  - echo $Var

- Reminder - echo prints to screen

  - man echo

  - optional: man printf

    - For fancier output

Adding Variables to script files

```
#! /bin/sh
STRING="HELLO WORLD" #assign variable
echo $STRING #prints the value
```

$IFS (Internal Field Seperator)

- This variable determines how Bash recognizes fields, or word boundaries, when it interprets character strings.
- $IFS defaults to whitespace (space, tab, and newline), but may be changed
- **echo "$IFS"** (With $IFS set to default, a blank line displays.)
- More details:
  - http://tldp.org/LDP/abs/html/internalvariables.html

Accessing Shell Script Arguments

- Positional parameters represent a shell script's command line arguments
- For historical reasons, enclose the number in braces if greater than 9

```
#! bin/sh
#test script
echo first arg is $1
echo tenth arg is ${10}

> ./argtest 1 2 3 4 5 6 7 8 9 10
```

If statements

- If statements use the test command or []
- man test
  - to see the expressions that you can create
  - e.g:

```bash
#!/bin/bash
if test "$#" -ne 2; then  #if number of input args not equal to 2 then...
   echo "Illegal number of parameters"
else
   if [ $1 -gt $2 ]; then  #if arg $1  >= arg $2 ...
      echo "1st argument is greater than 2nd"
   else
      echo "not possible"
   fi
fi
```

If statements

```
if condition
then
    statements-if-true-1
[ elif condition
then
    statements-if-true-2
... ]
[ else
  statements-if-all-else-fails ]
fi
```

Exit: Return value

Check exit status of last command that ran with `echo $?`

Value	Meaning
0	Command exited successfully
>0	Failure during redirection
1-125	Command exited unsuccessfully. The meanings
126	Command found, but file was not executable
127	Command not found
>128	Command died due to receiving a signal

Quotes

- Three kinds of quotes
  - Backticks ``

```
$ echo $PATH /bin:/usr/bin:/usr/X11R6/bin:

  /usr/local/bin
```

Basic Command Searching

- $PATH variable is a list of directories in which commands are found

```
$ echo $PATH /bin:/usr/bin:/usr/X11R6/bin:
  /usr/local/bin
```

Accessing Shell Script Arguments

- Positional parameters represent a shell script's command-line arguments

- For historical reasons, enclose the number in braches if it's greater than 9.

```
echo first arg is $1
echo tenth arg is ${10}
```

Simple Execution Tracing

- To get shell to print out each command as it's execute, precede it with "+"

- You can turn execution tracing within a script by using:

  `set -x`: to turn it on

  `set +x`: to turn it off

POSIX Built-in Shell Variables

Variable	Meaning
#	Number of arguments given to current process.
@	Command-line arguments to current process. Inside double quotes, expands to individual arguments.
*	Command-line arguments to current process. Inside double quotes, expands to a single argument.
- (hyphen)	Options given to shell on invocation.
?	Exit status of previous command.
$	Process ID of shell process.
0 (zero)	The name of the shell program.
!	Process ID of last background command. Use this to save process ID numbers for later use with the *wait* command.
ENV	Used only by interactive shells upon invocation; the value of $ENV is parameter-expanded. The result should be a full pathname for a file to be read and executed at startup. This is an XSI requirement.
HOME	Home (login) directory.
IFS	Internal field separator; i.e., the list of characters that act as word separators. Normally set to space, tab, and newline.
LANG	Default name of current locale; overridden by the other LC_* variables.
LC_ALL	Name of current locale; overrides LANG and the other LC_* variables.
LC_COLLATE	Name of current locale for character collation (sorting) purposes.
LC_CTYPE	Name of current locale for character class determination during pattern matching.
LC_MESSAGES	Name of current language for output messages.
LINENO	Line number in script or function of the line that just ran.
NLSPATH	The location of message catalogs for messages in the language given by $LC_MESSAGES (XSI).
PATH	Search path for commands.
PPID	Process ID of parent process.
PS1	Primary command prompt string. Default is "$ ".
PS2	Prompt string for line continuations. Default is "> ".
PS4	Prompt string for execution tracing with set -x. Default is "+ ".
PWD	Current working directory.

Arithmetic Operators

Operator	Meaning	Associativity
++ --	Increment and decrement, prefix and postfix	Left to right
+ - ! ~	Unary plus and minus; logical and bitwise negation	Right to left
* / %	Multiplication, division, and remainder	Left to right
+ -	Addition and subtraction	Left to right
<< >>	Bit-shift left and right	Left to right
< <= > >=	Comparisons	Left to right
= = !=	Equal and not equal	Left to right
&	Bitwise AND	Left to right
^	Bitwise Exclusive OR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND (short-circuit)	Left to right
||	Logical OR (short-circuit)	Left to right
?:	Conditional expression	Right to left
= += -= *= /= %= &= ^= <<= >>= |=	Assign ment opera tor s	Right to left

Exit: Return value

Value	Meaning
0	Command exited successfully.
> 0	Failure during redirection or word expansion (tilde, variable, command, and arithmetic expansions, as well as word splitting).
1-125	Command exited unsuccessfully. The meanings of particular exit values are defined by each individual command.
126	Command found, but file was not executable.
127	Command not found.
> 128	Command died due to receiving a signal.

Example

```
if grep pattern myfile > /dev/null
then
    ... Pattern is there
else
    ... Pattern is not there
fi
```

case Statement

```
case $1 in
-f)
    ... Code for -f option
    ;;
-d | --directory) # long option allowed
    ... Code for -d option
    ;;
*)
    echo $1: unknown option >&2
    exit 1 # ;; is good form before `esac', but not
  required
esac
```

for Loops

```
for i in atlbrochure*.xml
do
    echo $i
    mv $i $i.old
    sed 's/Atlanta/&, the capital of the South/' < $i.old > $i
done
```

while and until loops

while *condition*
do
    statements
done

until *condition*
do
    statements
done

break and continue

- Pretty much the same as in C/C++

Functions

- Must be defined before they can be used
- Can be done either at the top of a script or by having them in a separate file and source them with the "dot" (.) command.

Example

```
# wait_for_user --- wait for a user to log in
#
# usage: wait_for_user user [ sleeptime ]
wait_for_user ( ) {
    until who | grep "$1" > /dev/null
    do
        sleep ${2:-30}
    done
}
```

Functions are invoked the same way a command is

```
wait_for_user tolstoy          Wait for tolstoy, check every 30 seconds
wait_for_user tolstoy 60            Wait for tolstoy, check every 60 seconds
```

The position parameters ($1, $2, etc) refer to the function's arguments.
The return command serves the same function as exit and works the same way

```
answer_the_question ( ) {
    ...
    return 42
}
```