Software Construction Laboratory

Week 2

Lab 3

Mushi Zhou

Winter 2017

Prof. Paul Eggert

UCLA

Commands and Basic Scripting

Outline

- Unix wildcards & basic regular expressions
- Some advanced commands
- Piping and redirection
- Interpreted language
- Shell Scripting

Character Matching

Two types:

- Unix Wildcards (for all Unix commands)

  - man 7 glob

- Regular expressions (for Python Scripts/ other programmings)

  - man 7 regex

Unix Wildcards

- **?** Represent any *single* character
- ***** Represent any number of characters (including zero characters)
- **[ ]** Specifies a range.

  Ex: a[a,o,u]b -> aab, aob,aub   a[a-c]b -> aab, abb, acb.
- **{ }** Match terms. Separated by commas and each term can be a wildcard. No space between commas.
- **[!]** Logical NOT of []
- \ Used as an "escape" character, i.e. to protect a subsequent special c haracter.

  Ex: "\\\\" searches for a backslash.

Regular Expressions (Regex)

We will only focus on basic regular expressions

- . Match any single character, equivalent to ? in Unix wildcard
- \ Used as an "escape" character. Same in standard wildcard
- ***** The proceeding item is to be matched *zero or more* times
- **.*** Used to match any string, equivalent to * in standard wildcards

Regex

- **^** (caret) Means "the beginning of the line"

  Ex: "^a" means find a line starting with an "a"
- **$** Means "the end of the line".
- **[ ]** Specifies a range
- **|** This wildcard makes a logical OR relationship between wildcards
- **[^]** This is the equivalent of [!] in standard wildcards

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

Some Useful Advanced Commands

We will go over:

- grep
- comm
- tr

You will need to use/understand for assignment 2:

ln test diff cmp find ls

grep

- Searches input files for a pattern and outputs all matched lines
- Syntax: grep [OPTIONS] *pattern* [FILES]

Some useful options:

- **-F** Match using fixed strings
- **-f** *file* Use patterns from the given file, one per line
- **-i** Perform pattern matching in searches without regard to case
- **-n** Precede each output line by its relative line number in the file, each file starting at line 1
- **-v** Select lines not matching any of the specified patterns.

Examples of Using grep

- To find all uses of the word "Posix" (in any case) in file **text.mm** and write with line numbers:

```
grep -i -n posix text.mm
```

- To find all empty lines in the standard input:

```
grep ^$
grep -v
```

comm

- Compare two sorted files line by line and produce three-column outputs (1st column for lines unique to FILE1, 2nd column for lines unique to FILE2, 3rd column for line common to both files)
- Syntax: comm [OPTIONS] *FILE1 FILE2*

Some useful options:

- -1 do not output lines unique to file 1
- -2 suppress lines unique to file 2
- -3 suppress line common to both files
- - for FILE1 -> taking standard input as FILE1

tr

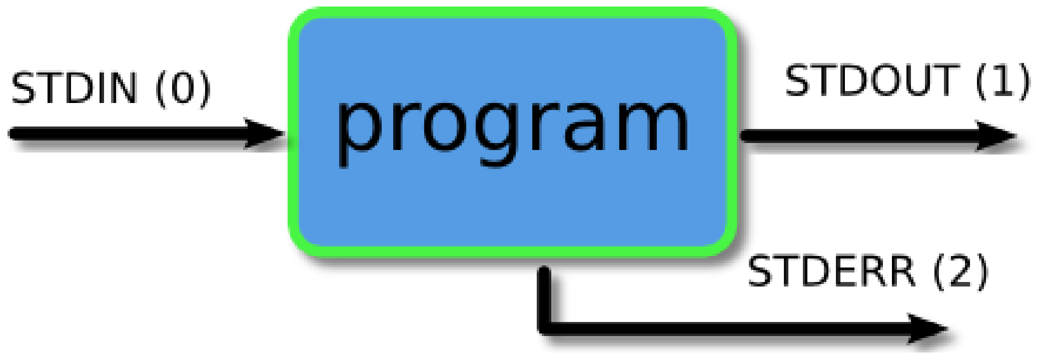
- Apply translation (or deletion) of characters from SET1 to SET2
- Syntax: tr [OPTIONs]... *SET1* [*SET2*]

Some useful options:

- -c use complement of SET1
- -s squeeze the repeated characters that are adjacent to each others
- -d delete characters in SET1, do not translate

See assignment 2 examples

Basic I/O Redirection



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

- STDIN (0) - Standard input (data fed into the program)
- STDOUT (1) - Standard output (data printed by the program, defaults to the terminal)
- STDERR (2) - Standard error (for error messages, also defaults to the terminal)

Redirection

- Use *program* < *FILE* to make *program*'s standard input be *FILE (i.e. taking inputs from FILE*
- Use program> *FILE* to make *program*'s standard output be *FILE (i.e printing to the FILE instead of terminal)*
- Use *program >> FILE* to append *program*'s standard output to the end of *FILE*.

- Redirecting STDERR*:   program 2> FILE*
- Redirecting STDERR to STDOUT*:   program 2>&1*

Piping

- Putting the output of a command to the input of another
- Combining multiple commands into one
- |

- `ls | head -3`
- `ls | head -3 | tail -1`

Interpreted Language v.s. Compiled Language

- Compiled Languages
  - Ex: C/C++, Java
  - Programs are translated from their original source code into object code that is executed by hardware
  - Can be optimized during compilation for specific systems

Interpreted Language

- Also known as Scripting Language
- Is a programming language for which most of its implementations execute instructions directly, without previously compiling a program into machine-language instructions
- The interpreter executes the program directly, translating each statement into a sequence of one or more subroutines already compiled into machine code
- May not be optimized for local system but there is not compilation time/process
- Example: Shell / Perl

Shell Script

- A shell script is a computer program designed to be run by the Unix shell
- An Interpreted Program
- Mostly a file made of commands
- Specifically for Unix-like systems, it’s different in windows (PowerShell, cmd.exe), DOS, etc.

Why Use a Shell Script

- Simplicity
- Portability
- Ease of development
- Commands and syntax are exactly the same as those directly entered at the command-line.
- Can be used to provide a sequencing and decision-making linkage around existing programs
- Interpretive running makes it easy to write debugging code into a script and re-run it to detect and fix bugs.

Disadvantages

- Prone to costly errors
- Slow Executions (Much slower for large executions)
- Difficult to write quality code

How to start writing a shell script

- The first line: #! /bin/sh
- This tells the kernel which shell to use
- There are other types of shells (i.e. dash, and python shell), but stick with sh for most of the course work
- Without this line, the script is not going to run
- While files with the ".sh" file extension are usually a shell script of some kind, most shell scripts do not have any filename extension
- You need to learn most of the syntax yourself

- We are using shell scripts to grade your assignments in this course!

Simple Example of Shell Script

```
#!/bin/sh

clear
ls –l
echo Hello World!
```

Shell Variables

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

- Examples:

• myvar=this_is_a_long_string_	Assign a value
• echo $myvar	Print the value
• first=hahaha last=test	Multiple assignments allowed on one line
• fullname=“hahaha test"	Use quotes for whitespace in value
• oldname=$fullname	Quotes not needed in value
• fullname="$first $middle $last"	Double quotes required for concatenating

Special Variables

- $0 The filename of the current script
- $n The input arguments. The first is $1, the second is $2, etc
- $#   The number of arguments supplied to a script
- $*   Arguments passed to the script, double quoted together

  i.e. If a script receives two arguments, $* is equivalent to $1 $2.
- $@ Arguments are individually double quoted. Similar to $*
- $?   The exit status of the last command executed
- $$ The process number of the current shell.
- $!   The process number of the last background command

Accessing Shell Script Arguments

- Enclose the number if it’s greater than 9

- Examples:

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

Example of Accessing Shell Script Arguments

Script:

```
#! /bin/sh
# scriptname
echo print arguments $1 $2
```

Run it:

```
$ chmod +x finduser                          Make it executable
$ ./scriptname first second
print arguments first second
```

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

Conditional Statements and Loops

- You can use conditional statements and loops in shell scripts
- i.e. if/elif/then/else/fi
- While/for/do/done
- These are very similar to python programs (with a little bit differences)
- Examples:

```
for i in 1 2 3; do
  echo "$i"
done
```

Some Help For Assignment 2

- Assignment 2 is now due Tuesday Midnight of next week!
- You are writing shell scripts for both homework and lab

Step Approach to Lab2

- Get the English dictionary “words”
- Spell check the assignment page
- Build a script “buildwords” that executes the rules mentioned in the lab
- Run script against “English to Hawaiian” page to form Hawaiian dictionary “hwords” after sorting the output
- Verify Hawaiian spell checker by running against itself
- Spell check the assignment page with “hwords” after ensuring all lower cases
- Log your findings

Some help for HW2

- Find duplicate files in a given directory
- Sort them and only keep one of each duplicates (Prefer “.”)
- Use hard link to replace others (Read what’s hard link)
- Only immediate files
- Only regular files
- Special character file names (space, *, -)
- Report Errors

Step Approaches

1. find all regular files and list the names of them
2. **run through the list and generate a list of group of duplicates**
3. Find files start with “.” in each group
4. Sort “.” files (or all files if there are not any “.”) in each duplicate groups
5. Replace duplicates with hard links to the first in each group

Useful Tips

- Find –type
- Find –maxdepth
- Diff/cmp
- Ln {s} {l}
- Sort is case sensitive by default
- Need to use regex (wildcards) to match names starting with “.”
- Pay attention to names with special characters (need to escape them)
- Nested loop