

CS 35L, Week 2, Fall 2017

Software Construction Laboratory

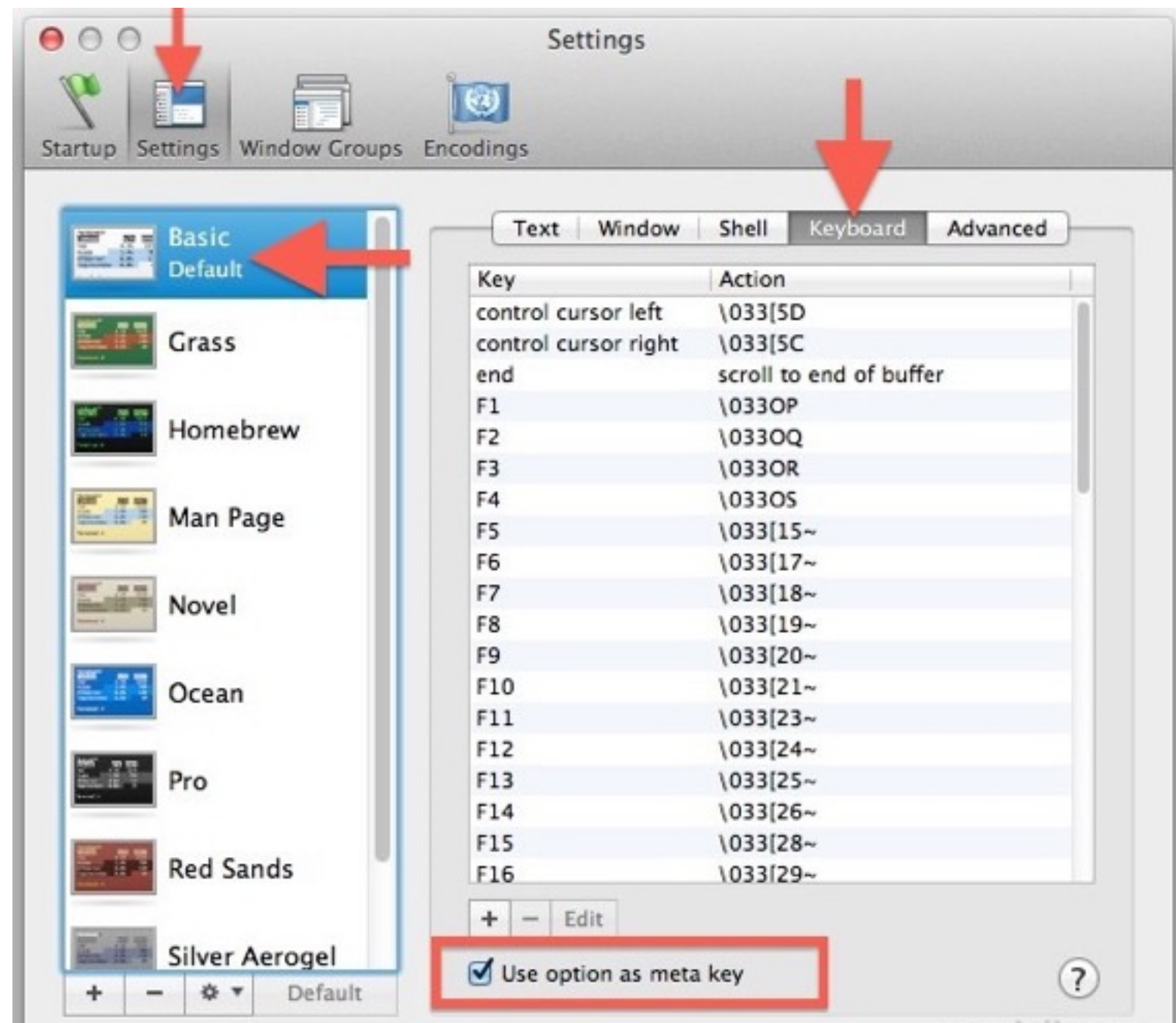
Moustafa Alzantot
CS Department, UCLA
malzantot@ucla.edu

Week1 followup

- Connecting to SEAS linux servers
 1. If outside the campus, you need to use VPN to connect to campus network.
 2. Use ssh to connect to one of the linux servers.
`ssh [username]@lnxsrv.seas.ucla.edu`
on Windows : use Putty
More instructions here:
<http://www.seasnet.ucla.edu/lnxsrv/>

Week1 followup

- to use options(alt) key as Meta key for EMACs on Mac:
 - Open Terminal and pull down the primary Terminal menu to choose “Preferences”



Lab2: Shell Scripting

Locale

A locale

- Set of parameters that define a user's cultural preferences
 - Language
 - Country
 - Other area-specific things

`locale` command

prints information about the current
locale environment to standard output

Environment Variables

- Variables that can be accessed from any child process

Common ones:

- **HOME**: path to user's home directory
- **PATH**: list of directories to search in for command to execute
- **USER**: the current username.
- **SHELL**: shell name

Environment Variables

- Printing environment variable value
`echo $VARIABLE`
- Changing environment variable value:
`export VARIABLE=...`

LC_* Environment Variables

`locale` gets its data from the LC_* environment variables

Examples:

- LC_TIME

Date and time formats

- LC_NUMERIC

Non-monetary numeric formats

- LC_COLLATE

Order for comparing and sorting

Locale Settings Can Affect Program Behavior!!

Default sort order for the `sort` command depends:

- `LC_COLLATE='C'`: sorting is in ASCII order
- `LC_COLLATE='en_US'`: sorting is case insensitive except when the two strings are otherwise equal and one has an uppercase letter earlier than the other.

Other locales have other sort orders!

The 'C' Locale

- The default locale
- An environment of “least surprise”
- Behaves like Unix systems before locales

sort, comm, and tr

sort: sorts **lines** of **text** files

- Usage: sort [OPTION]...[FILE]...
- Sort order depends on locale
- C locale: ASCII sorting

comm: compare two **sorted** files **line by line** prints to STDOUT what common lines and unique lines to each file.

- Usage: comm [OPTION]...FILE1 FILE2
- Comparison depends on locale

tr: translate **or** delete characters

- Usage: tr [OPTION]...SET1 [SET2]

Shell Scripting and Regular Expressions

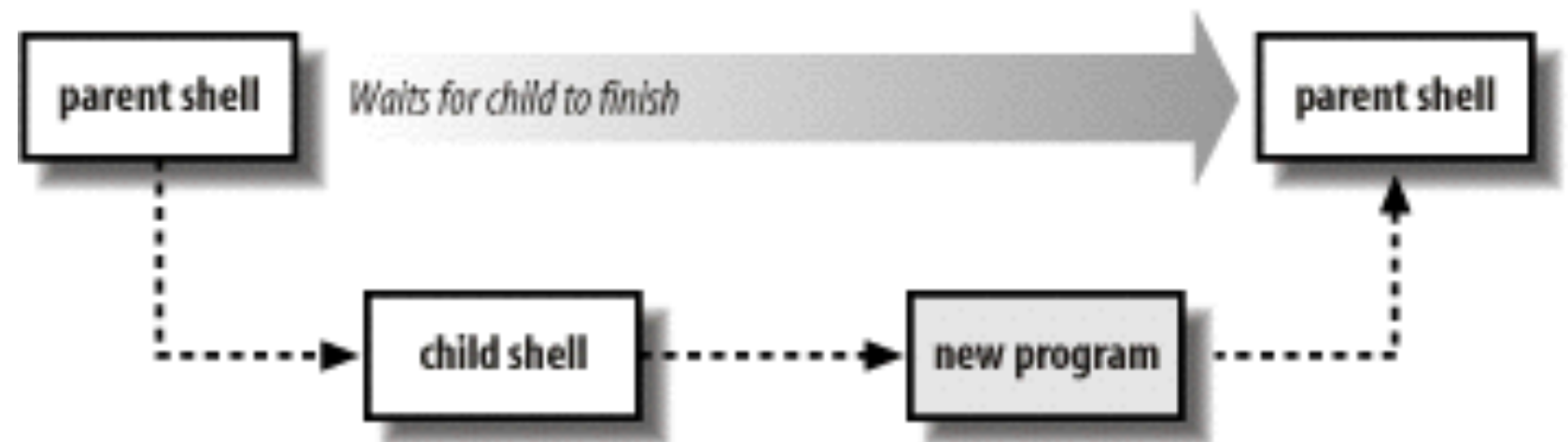
The Shell and OS

- The shell is a user interface to the OS
- Accepts commands as text, interprets them, uses OS API to carry out what the user wants – open files, start programs...
- Common shells
 - bash, csh, ash

Scripts: First Line

- A shell script file is just a file with shell commands
- When shell script is executed a new child “shell” process is spawned to run it
- The first line is used to state which child shell to use

```
#!/bin/sh  
#!/bin/bash
```



Compiled vs. Interpreted

Compiled languages

- Programs are translated from their original source code into machine code that is executed by hardware
- Efficient and fast
- Require recompiling
- Work at low level, dealing with bytes, integers, floating points, etc.
- Ex: C/C++

Interpreted languages

- Interpreter program (the shell) reads commands, carries out actions commanded as it goes
- Much slower execution
- Portable
- High-level, easier to learn
- Ex: PHP, Ruby, bash

Example

- An assignment directory for each lab

Before each lab:

- Create a new directory called “Lab02”
- Create 3 files in “Lab02”
 - lab.log
 - lab.txt
 - hw.txt

Debugging Tip: Execution Tracing

- Useful for debugging
- Shell prints out each command as it is executed
- Execution tracing within a script:
`set -x`: to turn it on
`set +x`: to turn it off

Output Using `echo` or `printf`

echo writes arguments to stdout, can't output escape characters (without `-e`)

```
$ echo "Hello\nworld"
```

```
Hello\nworld
```

```
$ echo -e "Hello\nworld"
```

```
Hello
```

```
world
```

By default `echo` will output a new line. Use `-n` to prevent new line at the end of the line.

printf can output data with complex formatting, just like C `printf()`

```
$ printf "%.3e\n" 46553132.14562253
```

```
4.655e+07
```

Variables

- Declared using =
 - `var="hello" #NO SPACES!!!`
- Referenced using \$
 - `echo $var`

```
#!/bin/sh
message="HELLO WORLD!!!"
echo $message
```


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 <i>wait</i> 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.

Exit: Return value

Check exit status of last command that ran with \$?

Value	Typical/Conventional Meaning
-------	------------------------------

0	Command exited successfully.
---	------------------------------

> 0	Failure to execute command.
-----	-----------------------------

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
-------	--

Accessing Arguments

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

```
#!/bin/sh
```

```
#test script
```

```
echo first arg is $1
```

```
#echo tenth arg is ${10}
```

```
./test hello
```

```
first arg is hello
```

if Statements

- If statements use the test command or []
- “man test” to see the expressions that can be done

```
#!/bin/bash
if [ 5 -gt 1 ]
then
    echo "5 greater than 1"
else
    echo "not possible"
fi
```


Quotes

- Three kinds of quotes
 - Single quotes `' '`
 - Do not expand at all, literal meaning
 - Try `temp='$hello$hello' ; echo $temp`
 - Double quotes `" "`
 - Almost like single quotes but expand backticks and `$`
 - Backticks `` `` or `$()`
 - Expand as shell commands
 - Try `temp=`ls` ; echo $temp`

Loops

- **While loop**

```
#!/bin/sh
COUNT=6
while [ $COUNT -gt 0 ]
do
    echo "Value of count is: $COUNT"
    let COUNT=COUNT-1
done
```

- The “let” command is used to do arithmetic

Loops

- **for loop**
 - f will refer to each word in `ls` output

```
#!/bin/sh
temp=`ls`
for f in $temp
do
    echo $f
done
```

Regular Expressions

- Notation that lets you search for text with a particular pattern:
 - For example: starts with the letter a, ends with three uppercase letters, etc.
- * Two versions of regular expressions: basic regular expressions (BRE), and extended regular expressions (ERE).

BRE is the default in sed and grep. Use '-r' or '-E' option to activate extended regular expressions.
- <http://regexpal.com/> to test your regex expressions
- Simple regex tutorial http://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 <code>\(...\)</code> and <code>\{...\}</code> .
.	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 <code>.</code> (dot) means any character, <code>.*</code> means "match any number of any character." For BREs, <code>*</code> 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 <i>interval expression</i> , 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 <i>holding space</i> . 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.

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., tolstoy, tolstoy, tolWHOttoy, and so on)

Regular Expressions (cont'd)

<code>\n</code>	BRE	Replay the nth subpattern enclosed in <code>\(</code> and <code>\)</code> into the pattern at this point. n is a number from 1 to 9, with 1 starting on the left.
<code>{n,m}</code>	ERE	Just like the BRE <code>\{n,m\}</code> 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.

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

POSIX Bracket Expressions

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

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: abcDEF def DEF
^[:alpha:]{3}	Characters 1, 2, and 3, at the beginning: abc ABCdefDEF

Searching for Text

- **grep: Uses basic regular expressions (BRE)**
“meta-characters `?`, `+`, `{`, `|`, `(`, and `)` lose their special meaning; instead use the backslashed versions” –`man grep`
- **egrep (or grep -E): Uses extended regular expressions (ERE) – no backslashes needed**
- **fgrep (or grep -F): Matches fixed strings instead of regular expressions.**

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)
```

sed

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

```
sed 's/regExpr/replText/ [g]'
```

- Example
 - Display the first directory in PATH
 - `echo $PATH | sed 's/:.*//'`

Remove everything after and including the first colon

- Display current directory name
 - `echo $PWD | sed 's/^\//.*\///'`

Text Processing Tools

- `wc`: outputs a one-line report of lines, words, and bytes
- `head`: extract top of files
- `tail`: extracts bottom of files

What is Lab 2 About?

Build a spelling checker for the Hawaiian language
(Get familiar with sort, comm and tr commands!)

- Steps:
 1. Download a copy of web page containing basic English-to-Hawaiian dictionary
 2. Extract only the Hawaiian words from the web page to build a simple Hawaiian dictionary. Save it to a file called hwords (**site scraping**)
 3. Automate site scraping: `buildwords` script (`cat hwnwdseng.htm | buildwords > hwords`)
 4. Modify the command in the lab assignment to act as a spelling checker for Hawaiian
 5. Use your spelling checker to check hwords and the lab web page for spelling mistakes

Lab Hints

- Run your script on seasnet servers before submitting to CCLE
- `sed '/patternstart/, /patternstop/d'`
 - delete patternstart to patternstop, works across multiple lines
will delete all lines starting with patternstart to patternstop
- The Hawaiian words html page uses `\r` and `\n` for new lines
 - `od -c hwnwdseng.htm` to see the ASCII characters
- You can delete blank white spaces such as tab or space using
 - `tr -d '[:blank:]'`
 - Use `tr -s` to squeeze multiple new lines into one
- `sed 's/<[^>]*>//g' a.html` to remove all HTML tags

*Send any question to
malzantot@ucla.edu*