Review

- What are regular expressions?
  - What can we represent with regular expressions?
  - How do we represent those things?
- Which command should you use for fast, enhanced searching with regular expressions?
- What are the benefits of bash scripts?
- How do we run bash scripts?
- What can we do in bash so far?

Today

- bash

Bash Scripting

- What are some programming language structures that we still need to learn about Bash?

Copy the $CS397/handouts/bash directory into your cs397 directory

Parameters

- A parameter is one of the following:
  - A *positional* parameter, starting from 0
  - A *special* parameter
- To get the value of a parameter: `${param}`
  - Can be part of a word (`abc${foo}def`)
  - Works within double quotes
- The `{}` can be omitted for simple variables, special parameters, and single digit positional parameters

Positional Parameters

- The arguments to a shell script
  - `$0, $1, $2, $3 …`
  - Parameter 0 is the name of the shell or the shell script
- The arguments to a shell *function*
- Arguments to the `set` built-in command
  - `set this is a test`
    - `$1=this, $2=is, $3=a, $4=test`
- Manipulated with `shift`
  - `shift 2`
    - `$1=a, $2=test`

Example with Parameters

Script

```
#!/bin/sh

# Parameter 1: string
# Parameter 2: file
grep $1 $2 | wc -l
```

Invocation:

```
$ bash countlines ing /usr/share/dict/words
30415
```

countlines

Special Parameters

Parameter	Meaning
$#	Number of positional parameters
$-	Options currently in effect
$?	Exit value of last executed command
$$	Process number of current process
$!	Process number of background process
$*	All arguments on command line from 1 on
"$@"	All arguments on command line Individually quoted "$1" "$2" …; good if parameters contain spaces

countlines_params

Special Characters

- The shell processes the following characters specially unless quoted:
  - `| & ( ) < > ; " ' $ ` space tab newline`
- The following are special whenever patterns are processed:
  - `* ? [ ]`
- The following are special at the beginning of a word:
  - `# ~`
- The following is special when processing assignments:
  - `=`

Here Documents

- Shell provides alternative ways of supplying standard input to commands (an anonymous file)
- Shell allows in-line input redirection using << called ***here documents***
- Syntax:

```
command [arg(s)] << arbitrary-delimiter
      command input
       :
       :
      arbitrary-delimiter
```

- `arbitrary-delimiter` should be a string that does not appear in text

Here Document Example

```
#!/bin/sh

mail –s "Groceries" sprenkles@wlu.edu << END
Don't forget your grocery list
Eggs
Milk
Bread
END
```

(Only works on hydros, which has the mail server.)

Command Substitution: ``

- Used to turn the output of a command into a string
- *Used to create arguments or variables*

```
$ date
Mon Jan 30 12:51:50 EST 2017
$ NOW=`date`
$ echo $NOW
Mon Jan 30 12:51:54 EST 2017
$ PATH=`myscript`:$PATH
```


Compound Commands

- Multiple commands
  - Separated by semicolon or newline
- Command groupings
  - pipelines
- Subshell

```
( command1; command2 ) > file
```

- Boolean operators
- Control structures

Boolean Operators

- Exit value of a program is a number
  - 0 means success
  - anything else is a failure code
- cmd1 && cmd2
  - executes cmd2 if cmd1 is successful
- cmd1 || cmd2
  - executes cmd2 if cmd1 is not successful

Send output to black hole (Can't be read)

```
$ ls bad_file > /dev/null && date
$ ls bad_file > /dev/null || date
Mon Jan 30 12:32:05 EST 2017
```


Control Structures

```
if expression
then
        command1
else
        command2
fi
```


What is an expression?

- Any UNIX command.
- Evaluates to true if the exit code is 0, false if the exit code > 0
- Special command **/bin/test** handles most common expressions:
  - String compare
  - Numeric comparison
  - Check file properties
- **[ ]** often a built-in version of **/bin/test** for syntactic sugar

Examples

```
if test $USER = "sprenkle"
then
        echo "I know you"
else
        echo "I don't know you"
fi
```

know.sh

```
if [ -f /tmp/stuff ] && \
    [ `wc –l /tmp/stuff | cut -f1 -d " "` -gt 10 ]
then
        echo "The file has more than 10 lines in it"
else
        echo "The file is nonexistent or small"
fi
```

filesize.sh

test Summary

- String based tests

`-z string`	Length of string is 0
`-n string`	Length of string is not 0
`string1 = string2`	Strings are identical
`string1 != string2`	Strings differ
`string`	string is not NULL

- Numeric tests

`int1 -eq int2`	First int equal to second
`int1 -ne int2`	First int not equal to second
`-gt, -ge, -lt, -le`	greater, greater/equal, less, less/equal

test Summary

- File tests

`-r file`	File exists and is readable
`-w file`	File exists and is writable
`-f file`	File is regular file (exists)
`-d file`	File is directory
`-s file`	File exists and is not empty

- Logic

`!`	Negate result of expression
`-a, -o`	And operator, or operator
`( expr )`	Groups an expression

What does this code do?

```
ARGS=1      # Number of arguments expected
# Exit value if incorrect number of args passed.
E_BADARGS=65

test $# -lt $ARGS && echo "Usage: `basename $0` <arg1>" && \
exit $E_BADARGS
```

- Add appropriate code to `countlines`

Arithmetic

- Use external command `/bin/expr`
- `expr expression`
  - Evaluates expression and sends the result to standard output
  - Yields a numeric or string result

```
expr 4 "*" 12
expr "(" 4 + 3 ")" "*" 2
```

Need to quote the * b/c shell interprets it

  - Particularly useful with command substitution

```
X=`expr $X + 2`
```

`arith_operators.sh`

Double parentheses and the `let` statement

- Double parentheses

```
z=$(($z+3))
z=$((z+3))
```

- Let statement

```
let z=z+3
let "z += 3"
```

Quotes permit the use of spaces in variable assignment

let.sh

Control Structures Summary

- `if … then … fi`
- `while … done`
- `until … do … done`
- `for … do … done`
- `case … in … esac`

`for` loops

```
for var in list
do
      command
done
```

- Examples:

```
sum=0
for var in "$@"
do
       sum=`expr $sum + $var`
done
echo "The sum is $sum"
```

sum_params.sh

```
for file in *.sh
do
      echo "We have $file"
done
```

for_file.sh
for_params.sh

Looking Ahead

- Assignment 2 due Wednesday