CSE 374: Programming Concepts and Tools

Spring 2026
Instructor: Megan Hazen

Today's Goals

Subject Matter

- Scripting Practice
- Intro to regular Expressions

Your Goals

- HW1

Aside: find things

```
[mh75@calgary ~]$ find -name shiftdemo
./cse374/shiftdemo
[mh75@calgary ~]$ whereis gcc
gcc: /usr/bin/gcc /usr/lib/gcc
/usr/libexec/gcc
/usr/share/man/man1/gcc.1.gz
/usr/share/info/gcc.info.gz
[mh75@calgary ~]$ which gcc
/usr/bin/gcc
```

- `man -k`: find commands with subject search (or apropos)
- `find`: location a file on a computer
- `locate`: locate a file in the directory database)
- `whereis`: finds files with a program's name
- `which`: where the executable in your path is found
- `diff f1 f2`: find lines that are different in f2 than in f1 (or sdiff)
- `grep`: find patterns in files

Source & Executable

We demonstrated source, but what about running a script?

- Want to run in a separate shell
- Need to specify what shell / interpreter
  - `#/bin/bash`
- Need to make the file executable
  - `chmod u+x`

```
CHMOD(1)

NAME
       chmod - change file mode bits

SYNOPSIS
       chmod [OPTION]...  MODE[,MODE]...  FILE...
       chmod [OPTION]...  OCTAL-MODE FILE...
       chmod [OPTION]...  --reference=RFILE FILE...

[mh75@calgary cse374]$ chmod 744 hello.sh
```

Gotchyas

- White space – it matters!
  - Assign WITHOUT spaces around the =
  - Bracket are WITH SPACES
- Typo on left creates a new variable
- Typo on right returns empty string
- Re-using a variable name changes the value
- Put quotes around values in case they have spaces in them
- Non-number string converts to a ‘0’

Shell Scripting

Bash Scripts

- Interpreted
- Esoteric variable access
- Everything is a string
- Easy access to files and processes
- Quick (to make & execute)
- Good for automating processes and building interactions

Java Programming

- Compiled
- Highly structured, strong typed variables
- String libraries
- Has data structures and libraries
- More overhead to write well
- Good for large, complex programs

A note on parenthesis etc.

Command substitution:
$(*command*) or `*command*`

Test:
test *condition* or [*condition*]

Upgrade: [[*condition*]]

Subshell: (*command*)

Math: ((*expression*))

Convert to string: $( ) or $(( ))

Tests:
-eq : equals.
-lt : Less than.
-e <file_a>: File_a exists.
-f <file_a>: File_a exists and a regular file.
-d <file_a>: File_a exists and is a directory.
-w <file_a>: File_a exists with write permissions.
-x <file_a>: File_a exists with execute permissions.

```
wget https://courses.cs.washington.edu/courses/cse374/26sp/lectures/demoscript
```

Demo

```
[mh75@calgary cse374]$ ./demoscript nospace 'with
space'
number of args: 2
name of script: ./demoscript
all args: nospace with space
all args: nospace with space
last exit: 0
$* value is nospace with space
$@ value is nospace
$@ value is with space
$var1+$var2 =  2+3
$(( $var1+$var2 )) =  5
let var3="$var1+$var2" =  5
[ Comparison ] worked
./demoscript: line 34: 5: No such file or
directory
[[ Comparison ]] worked
nospace with space
0 is nospace
1 is with space
2 is
3 is
```

1. Use wget to download the script
2. Run the script
    1. What do you have to do first?
    2. What happens if you call it with different arguments?
3. Add an exit command
4. Run the script again, and then query for whether it completed without error.
    1. Hint: echo `$?`

```
wget https://courses.cs.washington.edu/courses/cse374/26sp/lectures/fibo
```

Counting

```
#!/bin/bash
# this script returns the fibonacci sequence for the input
number

if [ $# -lt 1 ]; then
  # use default length
  limit=50
else
  # use argument
  limit=$1
fi

# seed the sequence
first=1; second=1

echo "Fibonacci sequence up to $limit"
echo $first; echo $second

while [ $second -lt $limit ]; do
  sum=$(( $first+$second ))
  echo $sum
  first=$second
  second=$sum
done
```

1. Use wget to download the script
2. Run the script
   1. What do you have to do first?
   2. What happens if you call it with different arguments?

```
wget https://courses.cs.washington.edu/courses/cse374/26sp/lectures/sdel
```

Utilities

```
[mh75@calgary ~]$ echo 'test file' > testingfile
[mh75@calgary ~]$ ls
cse374  testingfile
[mh75@calgary ~]$ sdel testingfile
Cleaning /data/netid/mh75/TRASH directory
[mh75@calgary ~]$ ls
cse374  TRASH
[mh75@calgary ~]$ ls TRASH/
testingfile.tar
```

1. There is a lot happening in this script.
   1. It checks for some system state
   2. It processes as many input files as we give it
   3. It calls other Bash utilities
   4. It uses some info about the file and the current date to choose an action
2. There is also something happening in my (Megan's) state:
   1. I modified by .bashrc file:
      ```
      # Adding path to the 374 scripts
      PATH=~/cse374/scripts:$PATH
      export PATH
      ```
   2. Now I can run any scripts that I keep in my scripts file without entering the path.

Practice

- https://courses.cs.washington.edu/courses/cse374/26sp/assignments/scriptchallenge.html

GREP

Run `man grep` now!

Grep matching

- Looks at one line at a time to match a pattern
- Default is to match .*p.*
  - (just means that anything can come before or after the string)
- But, you can anchor a pattern with ^ (beginning) and/or $ (end) or both.

```
[mh75@calgary scripts]$ ls -R ~ | grep demo
demoscript
shiftdemo
[mh75@calgary scripts]$ ls -R ~ | grep '^demo'
demoscript
[mh75@calgary scripts]$ ls -R ~ | grep 'demo$'
shiftdemo
[mh75@calgary scripts]$ ls -R ~ | grep '^demo$'
[mh75@calgary scripts]$
```

Regular Expressions

- A set of rules for matching a pattern (P) to a string (S)
- All strings are made of a combination of the null (empty) set, the empty string ε, and a single character.
- Regular expressions match a string if
  - P is a literal character (a, b, …) that matches the string S
  - P_1P_2 matches S if $S = S_1S_2$ such that P_1 matches S_1 and P_2 matches S_2
  - $P_1|P_2$ matches S if P_1 matches S OR P_2 matches S
  - P* matches S if there is an *i* such that P…P (*i* times) matches S. Includes *i=o* which matches ε.

RegEx rules

Regular expressions have

- Characters: the literal characters `[a b 9]` (S is an exact duplicate of P)
- Anchors: sets the position in the line where P may be found (`^` or `$`)
- Modifiers: modify the range of text P may match (`*` or `[set_of_chars]`)

Note: Regex details & implementation may vary between application, but general rules apply.

Is Bash 'globbing' the same as RegEx?

- Globbing: the shell filename expansion; matches some patterns
  - Essentially wild-card matching
  - Done by shell, outside the process
- Regular expressions (regex): a set of rules for matching patterns in text
  - Complext and powerful
- *(We see regular expressions in math, as formal grammars in cs, and other variations as well. Different applications (egrep) my have slightly different rules.)*