CSE 374: Programming Concepts and Tools

Spring 2026
Instructor: Megan Hazen

Today's Goals

Subject Matter

- Scripting
  - making executables `chmod`
  - File layout
  - Special variables & exit codes
  - Logic
  - Arithmetic
- Environment, scope, and export
- Style & Notes

Your Goals

- HW1
- Lecture 4 practice problem on Gradescope
- Process managements practice problem on Gradescope

Aside: passwd

Change your password (and do some other things like super users to manage other passwords) with

```
> passwd
```

- Linux systems have consistent password management

Well,
Calgary is actually a little different – passwords are obtained from UWNetID servers (not /etc/passwd entries).
Passwd **will** work and propagate change through the UWNetID servers….

Alias

- `alias`
  - Returns any aliases you have set
  - Defines a 'short-cut' or alias to a command
  - `unalias`

Aside - `.bashrc / .bash_profile`

'preferences' files that include your personal environment settings

`.bash_profile` is sourced every time you log-in

`.bashrc` is sourced every time you open a new interactive shell (usually called by `.bash_profile`)

Can put aliases or environment variables in `.bashrc` if you want to always use them.

Shell states & Scripting

- Shell **has a state**
  - Working directory, aliases, variables, history, streams
- Can change your state with command line operations (such as defining a new alias)
- Can use `source` to execute all the commands in a file, in your current shell, and **change your state**.
- You can also run commands in a file **without** changing state – a SCRIPT!

```
[mh75@calgary cse374]$ echo
"alias six='echo 6 7'" >>
party
[mh75@calgary cse374]$ six
-bash: six: command not
found
[mh75@calgary cse374]$
source party
[mh75@calgary cse374]$ six
6 7
```

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
```

DEMO

wget https://courses.cs.washington.edu/courses/cse374/26sp/lectures/hello.sh

Special variables for scripting

`$#` stores number of parameters (string

`$0` first string entered - the command

`$N` returns the Nth argument

`$?` Returns state of last exit

`$*` returns all the arguments

`$@` returns a space separated string with

(`*` returns one word with spaces,

`Shift` moves arguments to left (`$i` be

Note:
variables are strings
retain the value of something using 'single quotes'
Retain the value except `$, ` , \` use "double quotes"

Put `$*` and `$@` in quotes in case the arguments have spaces in them.

Exit codes

Processes in Bash exit with a numerical code

No error:

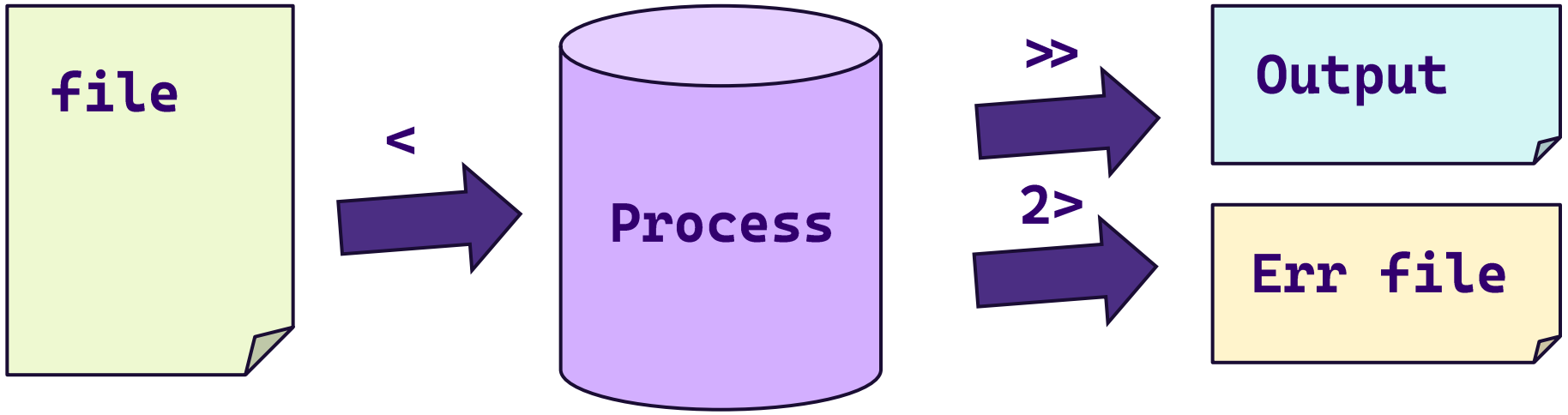
- `.exit`
- `.exit 0`

With an error:

- `.exit 1`
- or… `exit {1-255}`

IO Redirection

Special file: /dev/null
Is EoF if input
Data is discarded if output



- All processes have these three streams, but, you can redirect
  - Use '<' to redirect a file in
  - Use '>' to redirect out. '>>' appends, '2>' says redirect the second stream, or error

Redirection of script output

To redirect:	Use this syntax
`redirect stdout to a file`	*command* **`> output`**
`redirect stderr to a file`	*command* **`2> output`**
`redirect stdout to stderr`	*command* **`1>&2 output`**
`redirect stderr to stdout`	*command* **`2>&1 output`**
`redirect stderr and stdout to a file`	*command* **`&> output`**

Arithmetic

- Variables hold strings
- Math operators may not do what you want
- `K=$i+$j` concatenates!
- Shell function `((` performs math
  - `K=$(( $i+$j ))`
    - Strings converted to numbers
    - Non-numerical string converted to o
- You could also use
- Let `k="$i+$j"`

```
[mh75@calgary cse374]$ i=7
[mh75@calgary cse374]$ j=1
[mh75@calgary cse374]$ k=$i+$j
[mh75@calgary cse374]$ echo $k
7+1
[mh75@calgary cse374]$ k=$(( $i+$j ))
[mh75@calgary cse374]$ echo $k
8
[mh75@calgary cse374]$ let m=$i+$j
[mh75@calgary cse374]$ echo $m
8
```

Logic within scripts

Flow control stuff (as you expect)

```
if test; then
  commands
fi

while test; do
  commands
done

for variable in words; do
  commands
done
```

Conditionals

- Can use binary operators
  - `-eq -ne -lt -le -gt -ge`
- Can use unary operators
  - `-a file, -f regular_file, -r readable_file, -w writerable_file, -d directory`
- Can use shell command `[[`
  - And then `< > ==`

Logic example

Flow control stuff (as you expect)

```
if test; then
  commands
fi

while test; do
  commands
done

for variable in words; do
  commands
done
```

```
if [ -f ~/.bash_profile ]; then
    echo "You have a .bash_profile.
Things are fine."
else
    echo "Yikes! You have no
.bash_profile!"
fi

for (( c=1; c<=5; c++ ))
do
   echo "Welcome $c times"
done

for v in "$*"
do
    echo $v
    echo .
done
```

Environment, scope, export

- Variables in Bash are global *within the environment*
- If you want to use a variable within a child-process: `export myVar`
  - `export -n` to remove export property
  - Passes by value

Could declare a local value within a function

```
func1()
{
local var='func1 local'
}
```

Functions

- Yes!
  - Define with () and {}
  - Take input
  - Return a numerical value return code

```
cat is meow
dog is woof
fish+sauce
0
cat is
dog is woof
```

```
catfunction() {
    local cat="meow"
    dog="woof"
    echo "cat is $cat"
    echo "dog is $dog"
    echo $1+$2
    return 0
}

catfunction fish sauce
echo $?
echo "cat is $cat"
echo "dog is $dog"
```

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’

Style guidance

- Scripts should be short
  - < 200 lines?
  - DO ONE THING and DO IT WELL
- Always use spaces, not tabs, to indent
- Comment with a '#'
- https://google.github.io/styleguide/shellguide.html

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

Practice makes perfect

HW1

Practice problems

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.