PURDUE UNIVERSITY®

CS 25200: Systems Programming

Lecture 4: Bash Wrap-up
Operating Systems and UNIX

Prof. Turkstra

Announcements

- Easy for me to dive too deep. Let me know if you're suffocating.

Lecture 04

- I/O redirection
- Quotes
- grep
- Basic regular expressions
- Operating systems
- UNIX file system structure
- Processes and users

I/O redirection - reading

- To redirect input for a program or command,

  `< file` `n < file` n is the file descriptor

  `<< file` `n << file` n is the file descriptor
- Example:

```
mail jeff@purdue.edu < my_document
```

I/O redirection - writing

- We can redirect the output from a program or command too!
  - `> file` and `n > file` Redirect output to `file`
  - `>> file` and `n >> file` Appends output to `file`
  - `>| file` and `n >| file` Overrides the noclobber option, if set
  - `>& number` Redirects the output to file descriptor `number`

Examples

```
#! /bin/bash

exec 3< $1
exec 4> $2

I=0

while read <&3 Line; do
  echo "Line ${I}: ${Line}" >&4
done

exec 3>&-
exec 4>&-

exit 0
```

Trouble with quotes

- There are various kinds of quotes, and each one can mean something different in Bash.
  - ' The single forward quote character
  - " The double quote character
  - ` The back quote character
  - \ The backslash character (sometimes used to escape quotes)

The single forward quote '

- Must appear in pairs
- Protects all characters between the pair of quotes
- Ignores all special characters
- Protects whitespace

Single quote examples

```
Path='/b/cs252'
echo The path for cs252 is $Path
```

Displays:

```
The path for cs252 is /b/cs252
```

```
echo 'The path for cs252 is $Path'
```

Displays:

```
The path for cs252 is $Path
```

```
echo 'The book costs $2.00'
```

Displays:

```
The book cost $2.00
```

Wildcard * exception

```
#! /bin/bash
ls
echo *
echo '*'
yuk='*'
echo $yuk
echo '$yuk'
exit 0
```

Produces the following output:

```
Chap-1  TESTS  TV  memo  x  y
Chap-1 TESTS TV memo x y
*
Chap-1 TESTS TV memo x y
$yuk
```

The double quote "

- Must come in pairs
- Protects whitespace
- Does not ignore
  - Dollar signs - `$`
  - Back quotes - `` ` ``
  - Backslashes - `\`

Double quote example

```
Path="/b/bee264"
echo "The path for ee264 is $Path"
```

Yields:

```
The path for ee264 is /b/ee264
```

```
echo "The book cost \$2.00"
```

Yields:

```
The book cost $2.00
```

Wildcard * exception

```
#! /bin/bash
ls
echo *
echo "*"
yuk="*"
echo $yuk
echo "$yuk"
exit 0
```

Produces the following output:

```
Chap-1  TESTS  TV  memo  x  y
Chap-1 TESTS TV memo x y
*
Chap-1 TESTS TV memo x y
*
```

The back quote `

- Used to issue a UNIX command and obtain its output…

```
#! /bin/bash
echo "Current directory is `pwd`"
DIR=`pwd`
echo "Directory is ${DIR}"
exit 0
```

- Outputs:

```
Current directory is /a/turkstra/252
Directory is /a/turkstra/252
```

$(command)

- "Better" way to issue a UNIX command and obtain its output
  - Makes your code easier to read, easier to debug

```
#! /bin/bash
echo "Current working directory is $(pwd)"
DIR=$(pwd)
echo "Directory is ${DIR}"
exit 0
```

- Results in…

```
Current directory is /a/turkstra/252
Directory is /a/turkstra/252
```

The backslash `\`

- Used to remove any special meaning that a symbol may have. `\$1.00` "escapes" the `$` character, treating it as a literal `$` instead of `$1` (the first argument passed to the script)
- Used to add special meaning to symbols. `\n` for newline, for example.
- If it is the last symbol on a line, it will act as a continuation indicator.

Backslash example

```
#! /bin/bash
echo "This item costs \$2.00"
echo -n "This is line 1, "
echo "this is the rest of the line"
echo "This is" \
  "\"$(whoami)\"" # \" used to print "
exit 0
```

Outputs...

```
This item costs $2.00
This is line 1, this is the rest of the line
This is "turkstra"
```

Whitespace protection

```
#! /bin/bash
DATA=$(cat $0)
echo ${DATA}
echo
echo "${DATA}"
exit 0
```

Displays:

```
#! /bin/bash DATA=$(cat $0) print ${DATA} print print "${DATA}" exit 0
```

```
#! /bin/bash
DATA=$(cat $0)
print ${DATA}
print
print "${DATA}"
exit 0
```

`grep` command

- Used to search files for lines of information. Many, many flags - see the man page.
  `grep -flags regular_expression filename`
- Useful flags...
  `-x` Exact match of line
  `-i` Ignore upper/lower case
  `-c` Only count the number of lines which match
  `-n` Add relative line numbers
  `-b` Add block numbers
  `-v` Output all lines which do not match

Simple regular expressions

- Regular expressions express patterns. They are used to find and/or extract pieces of information from a string.
  - `.` Matches any character
  - `^` Start of line
  - `$` End of line
  - `\` Escape character
  - `[list]` Matches any character in the list
  - `[^list]` Matches any character not in the list
  - `*` Match zero or more occurrences of the previous regular expression
  - `\{min,max\}` Matches at least min and at most max occurrences of the previous regular expression

Examples

- `grep "^string$" file_name`
  collects all lines which contain only string
- `grep " … " file_name`
  collects all lines which have any three characters surrounded by spaces
- `grep " [0-9]\{1,3\} " file_name`
  collects all lines containing a sequence of one to three digits surrounded by spaces
- `grep "^x*[abc]" file_name`
  collects all lines which start with zero or more `x`'s followed by a single `a`, `b`, or `c`

More examples

- Let's pretend we have a file named `data`...

```
12
12 345
567
3 abd
asdf
```

- And this script...

```
#! /bin/bash                    # begins with 1 or 2
grep "^[0-9]\{1,2\} " data      # digits followed by
exit 0                          # a space
```

- We should get this output...

```
12 345
3 abd
```

-v option, inverting the match

- Let's pretend we have a file named `data`...

```
12
12 345
567
3 abd
asdf
```

- And this script...

```
#! /bin/bash
grep -v "^[0-9]\{1,2\} " data
exit 0
```

- We should get this output...

```
12
567
asdf
```

-c option, counting the matches

- Let's pretend we have a file named `data`...

```
12
12 345
567
3 abd
asdf
```

- And this script...

```
#! /bin/bash
grep -c "^[0-9]\{1,2\} " data
exit 0
```

- We should get this output...

```
2
```

-n option, adding line numbers

- Let's pretend we have a file named `data`...

```
12
12 345
567
3 abd
asdf
```

- And this script...

```
#! /bin/bash
grep -n "^[0-9]\{1,2\} " data
exit 0
```

- We should get this output...

```
2:12 345
4:3 abd
```

Using grep inside a script

```
#! /bin/bash
if (( $# != 1 )); then
  echo "Usage: $0 <user_id>"
  exit 1
fi
USER="$1"
if grep "${USER}" Id_File > /dev/null
then
  echo "Bad way: ${USER} in file"
else
  echo "Bad way: ${USER} not in file"
fi
if grep "^${USER}$" Id_File > /dev/null
then
  echo "Good way: ${USER} in file"
else
  echo "Good way: ${USER} not in file"
fi
exit 0
```

Output

```
$ cat Id_File
sam
maryann
john
jeff
jeffrey
bill
william
peterson
```

```
$ Check
Usage: Check <user_id>
$ Check jeff
Bad way: jeff in file
Good way: jeff in file
$ Check son
Bad way: son in file
Good way: son not in file
```

Bash and Regular Expressions

- Bash can match against regular expressions internally:

```
# !/bin/bash

echo -n “Feed me: “
read IN

if [[ “${IN}” =~ [0-9]{1,3} ]]; then
  echo “Match!”
else
  echo “Nope!”
fi
```

Purdue trivia

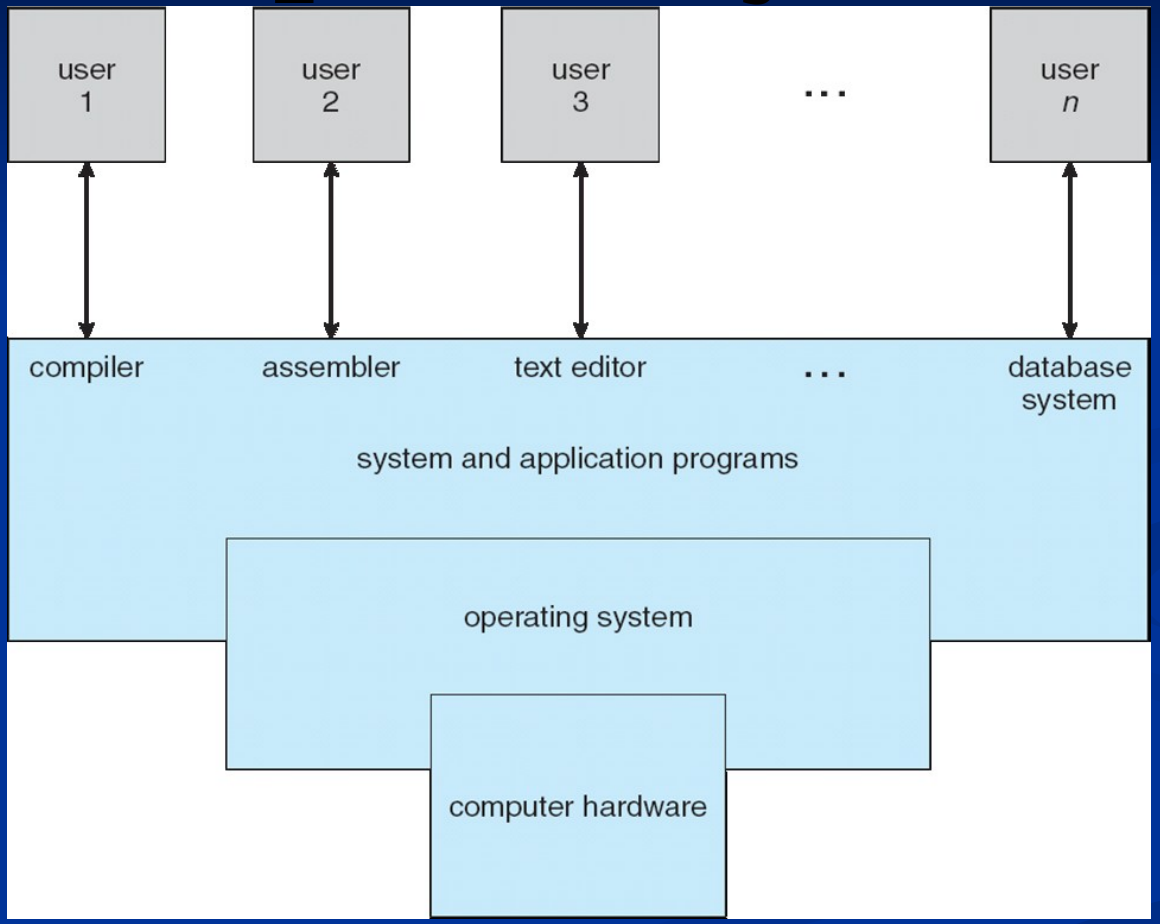
- "The first dean of agriculture, John Skinner, did some of Purdue's most effective lobbying in the Indiana General Assembly, armed with Purdue enthusiasm and a bushel or two of ripe apples from a university orchard."

  - A Century and Beyond, by Robert W. Topping

What is an operating system?

- Code! Code that...
  - Sits between programs and hardware
  - Sits between different programs
  - Sits between different users
- What does it do?
  - Manage resources
  - Extends (abstracts) the machine
  - Consumes resources
  - **Makes computers simpler**

Four components of a computer system



What is an OS?

- No universally accepted definition
- “Everything a vendor ships when you order an operating system” is a good approximation
  - Still varies wildly
- “The one program running at all times on the computer” is the kernel
- Everything else is either
  - A system program
  - An application program

Some features/software…

- Multitasking – multiple processes running in the same computer
- Multiuser – multiple users using the same computer
- File system(s) – interface to storage
- Networking – access to networks (like the Internet)
- Windowing system – graphical user interface

- Standard programs – web browser, task manager, editors, compilers, etc
- Common libraries – libraries used by many programs (math, string, window, standard C, etc)

Tour of UNIX

- Multics – an experimental, time sharing operating system for the GE-645 mainframe
  - Bell Labs slowly withdrew support
- Ken Thompson, Dennis Ritchie, Brian Kernighan, and others decided to ”redo it” in 1969
  - Smaller, faster, and more reliable

- “What we wanted to preserve was not just a good environment in which to do programming, but a system around which a fellowship could form. We knew from experience that the essence of communal computing, as supplied by remote-access, time-shared machines, is not just to type programs into a terminal instead of a keypunch, but to encourage close communication.” – Dennis Ritchie

Some characteristics

- The creators and programmers used UNIX to develop .. UNIX.
  - “Eating your own dog food”
  - Started out as UNICS – Uniplexed Information and Computing Service
- Computing Sciences Research Center and a desire for word processing resulted in funding to port it to a DEC PDP-11/20
  - At this point it became officially UNIX

- UNIX was (eventually) re-written in “C” (95%) and assembly language (5%)
- Relatively easy to port it to other machines

BSD UNIX

- UNIX was a success at universities and elsewhere
- People began writing their own versions
  - Berkeley – BSD-UNIX
    - Sockets, FTP, Mail, etc came from BSD
  - SunOS
  - Xenix (MS/SCO)
- POSIX was formed to create a standard that would permit interoperabilty amongst the *NIXes
  - Portable Operating System Interface

UNIX File System

- *NIX has a hierarchical file system
- Root of entire tree is denoted with /
- Disks and network shares can be mounted anywhere inside the hierarchy

Hierarchy

- /bin – historically contained fundamental utilities (ls, cp, etc)
  - Often now a symlink to /usr/bin
- /usr - “user file system”
  - Used to be split off on separate storage, usually not anymore
- /boot – files necessary for startup (e.g., initial kernel image)

- /dev – peripheral and other devices
- /tmp – temporary files that do not survive a reboot
- /var – files that may change frequently (variable)
  - Email, logs, databases, etc
- /sbin - “system (superuser) binaries”
  - Utilities needed to start and maintain/recover the system
  - Also symlinked to /usr/sbin

- /etc – system-wide configuration files
  - Init system scripts
    - /etc/rc.d
    - /etc/rc.*
  - User and group information
    - /etc/passwd, /etc/groups
- /home – user home directories
- /lib – essential libraries
  - Often split into lib, lib64 now
  - Also usually symlinked into /usr/lib*

- /usr/include – header files for libraries and kernel
- /proc – virtual file system that provides information about processes through files
- /sys – other hardware and kernel-related information

Processes

- top, ps (-e, -ax, -f, -u)

Users

- UNIX was designed as a multiuser system
- Database of users exists in /etc/passwd
  - ...but not for Purdue machines (NIS/LDAP)
  - Each user has a unique uid
  - Passwords are now stored in /etc/shadow

- useradd – create a new user
- passwd – change a user's password
- There exists a special user called "root"
  - Root can do anything
    - Modify files, change permissions/ownership, mount/umount, rm anything, execute anything, etc etc
  - To become root, you use the "su" command. (super user)

Got root?

- Many times systems are configured to use “sudo”. Sudo allows regular users to run certain (specified) things as root. See /etc/sudoers
  - Does not require the root user’s password, unlike su
  - Can “sudo su” to get a root shell, if you have the right privileges
- Root shells are dangerous
  - `$ rm -rf / home/turkstra/somefile`

Groups

- A group is a collection of users
- Users may belong to multiple groups
  - “Supplemental groups”
  - usermod -S, groupadd, groupmod
  - /etc/group

Network Information Service (NIS)

- Formerly Yellow Pages or YP
- Client-server directory service protocol for distributing information like users and groups
- See “ypcat”

Questions?