Principles of Programming
CM10227

Lecture S.2.: Introduction to Linux/Unix Part 2

Dr. Marina De Vos
University of Bath
Ext: 5053

Academic Year 2012-2013

Outline

Outline

Resources

- Unix for Beginners. Dirk Vermeir
- `http://osl.iu.edu/~lums/swc/www/index.html`

Outline

Previous Commands

pwd	rm	cat
ls	chmod	whoami
cd	wc	passwd
mkdir	more	users
cp	less	groups
mv	diff	

Redirecting Input and Output

- A running program is called a process
  - By default, every process has three connections to the outside world:
    - Standard input (stdin): connected to the keyboard
    - Standard output (stdout): connected to the screen
    - Standard error (stderr): also connected to the screen (Used for error messages)

Redirecting Input and Output II

You can tell the shell to connect standard input and standard output to files instead

- command < inputFile reads from inputFile instead of from the keyboard
  - Don't need to use this very often, because most Unix commands let you specify the input file (or files) as command-line arguments
- command > outputFile writes to outputFile instead of to the screen
  - Only normal output goes to the file, not error messages
- command < inputFile > outputFile does both

Redirecting Input and Output III

- Example: save number of words in all text files to words.len

```
\$ wc -w *.txt > words.len
```

- Nothing appears, because output is being sent to the file words.len

```
\$ ls -t
words.len       venus.txt       earth.
    txt
\$ cat words.len
 9  earth.txt
 9  venus.txt
 12 words.len
 30 total
```

Try typing cat > junk.txt

- No input file specified, so cat reads from the keyboard
- Output sent to a file
- Voila: the world's dumbest text editor

Pipes I

- Suppose you want to use the output of one program as the input of another
  - E.g., use wc −w ∗.∗ to count the words in some files, then sort −n to sort numerically
- Option 1: send output of first command to a temporary file, then read from that file

```
wc *.txt > temp
sort -n < temp
```

Pipes II

- Option 2: use a pipe to connect the two programs
  - Written as "|"
  - Tells the operating system to send what the first program writes to its stdout to the second program's stdin

```
wc -w *.* | sort -n
9 earth.txt
9 venus.txt
12 words.len
30 total
```

- More convenient (and much less error prone) than temporary files

Pipes III

- Can chain any number of commands together
  - And combine with input and output redirection
  - ```
    wc *.txt | sort -n | head -5 > shortest.files
    ```
- Any program that reads from standard input and writes to standard output can use redirection and pipes
  - Programs that do this are often called filters
  - If you make your programs work like filters, you can easily combine them with others
  - A combinatorial explosion of goodness


    ```

Environment Variables

- Like any other program, the shell has variables
- Since they define a user's environment, they are usually called environment variables
- Usually all upper case
- Type set at the command prompt to get a listing: (a cut down version)

```
\$ set
BASH=/bin/bash
HOME=/home/mdv
HOSTNAME=alis-tablet1
```

Environment Variables II

- Get a particular variable's value by putting a "$" in front of its name
  - E.g., the shell replaces "$HOME" with the current user's home directory
  - Often use the echo command to print this out

```
$ echo $HOME
/home/mdv
```

  - Question: why must you type echo $HOME, and not just $HOME?
- To set or reset a variable's value temporarily, use this:

```
\$ export VARNAME=value
```

  - Only affects the current shell (and programs run from it)

Environment Variables III

- To set a variable's value automatically when you log in, set it in /.bashrc
  - Remember, " " is a shortcut meaning your home directory

Environment Variables IV

Name	Typical Value	Notes
DISPLAY	0:0	The display variable used for X11 graphics.
HOME	/home/mdv	The current user's home directory
HOMEDRIVE	C:	The current user's home drive (Windows only)
HOSTNAME	"alis-tablet"	This computer's name
HOSTTYPE	"i486"	What kind of computer this is
OSTYPE	"linnux-gnu"	What operating system is running
PATH	"/usr/local/sbin: /usr/local/bin: /usr/sbin:/usr/bin: /sbin:/bin:/usr/bin/X11: /usr/games"	Where to look for programs
PWD	/home/mdv/Desktop/	Present working directory
SHELL	/bin/bash	What shell is being run
TEMP	/tmp	Where to store temporary files
USER	"mdv"	The current user's ID

How the Shell Finds Programs

- The most important of these variables is PATH
  - The search path that tells the shell where to look for programs
  - When you type a command like tabulate, the shell:
    - Splits $PATH on colons to get a list of directories
    - Looks for the program in each directory, in left-to-right order
    - Runs the first one that it finds
- Example
  - PATH is /home/mdv/bin:/usr/local/bin:/usr/bin:/bin:/Python24
  - Both /usr/local/bin/tabulate and /home/mdv/bin/tabulate exist
  - /home/mdv/bin/tabulate will be run when you type tabulate at the command prompt
  - Can run the other one by specifying the path, instead of just the command name

How the Shell Finds Program

- Warning: it is common to include . in your path
  - This allows you to run a program in the current directory just by typing whatever, instead of ./whatever
  - But it also means you can never be quite sure what program a command will invoke
  - Though you can use the command which program_name, which will tell you
- Common entries in PATH include:
  - /bin, /usr/bin: core tools like ls (Note: the word bin comes from binary, which is geekspeak for a compiled program)
  - /usr/local/bin: optional (but common) tools, like the gcc
  - $HOME/bin: tools you have built for yourself

Basic Tools I

man	Documentation for commands.
cat	Concatenate and display text files.
cd	Change working directory.
chmod	Change file and directory permissions.
clear	Clear the screen.
cp	Copy files and directories.
date	Display the current date and time.
diff	Show differences between two text files.
echo	Print arguments.
env	Show environment variables.
head	Display the first few lines of a file.
ls	List files and directories.
mkdir	Make directories.

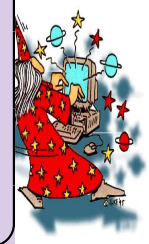
Basic Tools II

more	Page through a text file.
mv	Move (rename) files and directories.
od	Display the bytes in a file.
passwd	Change your password.
pwd	Print current working directory.
rm	Remove files.
rmdir	Remove directories.
sort	Sort lines.
tail	Display the last few lines of a file.
uniq	Remove duplicate lines.
wc	Count lines, words, and characters in a file.
which	locate a command

Compressing and Decompressing

- If large files have to be stored, emailed or transported, it is easier to compress them, making the file size smaller.
- When needed later they can be restored or decompressed
- A variety of compressing/decompressing algorithms exists.
- A simple example: the sequence ”aaaaa” could for example be represented more compactly as ”5a”
- The most common in linux systems are ”gzip - gunzip”,”bzip2 - bunzip2” and ”zip”, which gives you the extensions ”.gz”,”.bz2”,”.zip”

```
-rw-r--r-- 1 mdv mdv 514826 2007-10-11 00:01 lecture2.pdf
gzip lecture2.pdf
gunzip lecture2.pdf.gz
-rw-r--r-- 1 mdv mdv 382068 2007-10-11 00:01 lecture2.pdf.gz
bzip2 lecture2.pdf
-rw-r--r-- 1 mdv mdv 380635 2007-10-11 00:01 lecture2.pdf.bz2
bunzip2 lecture2.pdf.bz2
```



Creating Archives

- For storing purposes, emailing or transporting a number of files, it is sometimes more convenient to be able to bundle all these files together as one big file. Such a file is called an archive
- In linux we use a program called ”tar”, so we call those archives tarfiles.
- Format: tar –flags pathname [pathname]
- Common flags:

c	create an archive
x	extract an archive
t	display what is an archive
v	verbose, display the files being dealt with
f	use filename for archive
z	filter through gzip/gzunip
j	filter through bzip2/bunzip2

Creating Archives: Some examples

- `tar -cvfz myfiles.tar.gz lectures/`: creates an archive called myfiles.tar which is also being compressed using gzip and will include the directory lectures and all files that are in the directory lectures.
- `tar -tvf myfiles.tar`: this command will list all files that are stored in the archive myfiles.tar
- `tar -xvfz myfiles.tar.gz`: this will command will decompress the file myfiles.tar.gz and will then extract the archive in the current directory. This will result in a directory lectures being created (if this directory does not exist) where all the files of archive are going to be put.

More Advanced Tools II

du	Print the disk space used by files and directories.
bzip	Compress a file
bunzip	Decompresses a file
find	Find files that match a pattern.
grep	Print lines matching a pattern.
gunzip	Uncompress a file.
gzip	Compress a file.
lpr	Send a file to a printer.
lprm	Remove a print job from a printer's queue.
lpq	Check the status of a printer's queue.
ps	Display running processes.
tar	Archive files.
which	Find the path to a program.
who	See who is logged in.
xargs	Execute a command for each line of input.

The Last Linux Slide: Making life Easier

- When halfway typing a command on the command-line, try pressing the tab and see what happens.
- After typing and executing a command on the command-line, try the arrow keys and see what happens

Outline

Scope

```
x = 7

def scopetest(param):
    x = param
    print x
    if x == 0:
        return "yes"
    elif (x % 2) == 0:
        x = param / 2 - 1
        scopetest(x)
    else:
        x = x + 1
        scopetest(x)

scopetest(x)
```

```
7
8
3
4
1
2
0
```

Scope

```
x = 7

def scopetest(param):
    x = param
    print x
    if x == 0:
        return "yes"
    elif (x % 2) == 0:
        x = param / 2 - 1
        scopetest(x)
    else:
        x = x + 1
        scopetest(x)

scopetest(x)
```

```
7
8
3
4
1
2
0
```

How would you start on these?

1. Write a program that implements multiplication without using the use of *
2. Write a program that finds all prime dividers of a given number
3. Python has a module called random with a function random() providing random numbers between 0.0 and 1.0. How can you get numbers between 1 and 6 inclusive

How would you start on these?

1. Write a program that implements multiplication without using the use of *
2. Write a program that finds all prime dividers of a given number
3. Python has a module called random with a function random() providing random numbers between 0.0 and 1.0. How can you get numbers between 1 and 6 inclusive

How would you start on these?

1. Write a program that implements multiplication without using the use of *
2. Write a program that finds all prime dividers of a given number
3. Python has a module called random with a function random() providing random numbers between 0.0 and 1.0. How can you get numbers between 1 and 6 inclusive

And on these?

1. Given an ordered tree containing numbers, how would you go about finding a given number?
2. How would you go about writing a program that implements the game of Yahtzee?
3. Challenge! Global variables are not always a good idea. Everybody can access them. Suppose you want to limit access to a particular variables in such a way that can accessed in way you as a programmer intended.

And on these?

1. Given an ordered tree containing numbers, how would you go about finding a given number?
2. How would you go about writing a program that implements the game of Yahtzee?
3. Challenge! Global variables are not always a good idea. Everybody can access them. Suppose you want to limit access to a particular variables in such a way that can accessed in way you as a programmer intended.

And on these?

1. Given an ordered tree containing numbers, how would you go about finding a given number?
2. How would you go about writing a program that implements the game of Yahtzee?
3. Challenge! Global variables are not always a good idea. Everybody can access them. Suppose you want to limit access to a particular variables in such a way that can accessed in way you as a programmer intended.