# Unix/Linux commands and shell programming

*Clemson University*

*PARL*

Presented by Tim Shelling, UNIX guru

# UNIX Overview

Why UNIX?

- Control
  - Commands often provide complete access to the system and its devices
  - Most devices can be manipulated just like files
- Flexibility
  - Commands are just programs
  - Commands have common interface to allow interoperation with other commands
  - The UNIX shells provide the “glue” for this `
- Reliability
  - Commands are typically *lightweight* since they typically do little more than invoke operating system calls
  - Individual commands that are broken can easily be replaced
- *Summary:* All the above translate into…

***POWER***

# *UNIX: The Command Line*

- Accessing UNIX through a *terminal*
  - *telnet [hostname] [port]*
    - The omnipresent failsafe.  Nowadays, turned off due to lack of adequate security.
  - *ssh [user@]hostname*
    - *Secure. Data is encrypted over "the wire".* ***What we use.***
    - ***Not always available outside CU*** *due to different versions, implementations, platform availability.*
- Log in!

  3 tries to get valid username and password right
- Show who is logged in
  - `w` *or* `who`
  - `finger`
- Logout!
  - *exit*
  - CTRL-D

# UNIX: Accessing Documentation

- Commands are generally documented using the command *man*.
  - *man* pages are subdivided into various *sections*
  - Example: Documentation of the *man* command

    ```
    man man
    ```
  - Example: Documentation of the time *command*

    ```
    man time
    ```
  - Example: Documentation of the time *C library function*

    ```
    man 3 time
    ```
- *man* will present the manual page of the specified entry using *more* or *less.*
  - In Linux, the default is *less*, but can be overridden
  - *less* presents a screen-full at a time. ‘spacebar’ moves forward, ‘b’ moves backward, ‘$’ moves to end, ‘q’ quits, ‘?’ helps.

# UNIX: Accessing Documentation

- A few commands (such as diff, gcc, awk) are doccumented using *info*.
  - *info* is GNU-specific
  - Uses its own hypertext ‘viewer’.
    - arrow-keys select different links
    - *space* pages forward
    - *‘u’* goes back “up” a hyperlink level, like “back” in browsers
- Most commands have HTML references on the WWW.
- Don’t panic.  Just e-mail me or Dan.

# UNIX terminal management: **screen**

| | |
|---|---|
| Help | CTRL-A ? |
| Copy/Scrollback | CTRL-A [ |
| Paste | CTRLA ] |
| Lock | CTRL-A x |
| Detach | CTRL-A d |
| New Screen | CTRL-A c |
| Next/Previous | CTRL-A n / CTRL-A p |
| Reattach | screen –D –A –r |
| List active | screen -ls |

# UNIX: Getting around the filesystems

- UNIX files are organized just like they are with PC's and MAC's
  - Files are contained in collections of *Directories.*
  - Directories may contain other Directories
  - Different drives are "mounted" onto directories – **there are no drive letters!!**
  - The "top level" directory is called the "root" directory and is referred to by "/"
  - The current directory is referred to by "."
  - The directory one level up is referred to by ".."
  - More dots don't get you more levels up. ☹
  - *Shortcuts* in Windows are called *soft-links.* Act just like normal files, directories, or whatever it is they refer to.
  - Other filetypes include named pipes, character devices, block devices, sockets.

# UNIX: Getting Around

- Commands to navigate the directories:
  - pwd
  - ls
    ```
    ls file; ls directory ;  ; ls –a ; ls –l ; ls -R
    ```
  - cd
    ```
    cd ..
    cd /home/tim/projects
    cd ~/projects
    cd ~tim/projects
    cd $HOME/projects
    ```
  - mkdir
  - rmdir
  - mv
    ```
    mv oldfilename newfilename
    mv file1 file2 file3 newtargetdirectory
    ```
  - cp  -- syntax like mv
    ```
    cp –r dir1 dir1copy
    ```
  - rm
  - push
  - pop
  - find
    ```
    find . –ls
    find . –type d –print
    find . –type f –exec “echo” “{}” “;”
    ```

# UNIX: More Standard Commands

```
echo      print out a string
  echo “$HOME is where I want to be”
cat   Output specified files in sequence
  cat file1 file2 file3
whereis      Show where a file can be found
printenv     Display all environment variables
grep      Get Regular Expression and Print
head      first few lines of output
  head -5
tail      last few lines of output
  tail -8
```

# UNIX command line tricks

- Shell “glob”

```
# mkdir /tmp/moved
# mv * /tmp/moved
# cp /tmp/moved/* .
```

- Filename Completion (tcsh, bash)

  # ls /tmp/m<TAB>

- Command line history (tcsh)
  - history
  - CTRL-P and CTRL-N, down/up arrows
  - !previous – Runs the previous command beginning with the word *previous*.

# UNIX: The SHells

- The “Shell” is simply *another program* which provides a basic human-OS interface.
- Shells can run interactively or as a shell *script*
- Two main ‘flavors’ of Shells:
  - Bourne created what is now known as the *standard* shell: “sh”, or “bourne shell”. It’s syntax roughly resembles Pascal. It’s derivatives include “ksh” (“korn shell”) and now, the most widely used, “bash” (“bourne shell”).
  - One of the creators of the C language implemented the shell to have a “C-programming” like syntax. This is called “csh” or “C-shell”. Today’s most widely used form is the very popular “tcsh”.

# Unix: SH basics

- Modifying *environment variables*
  *sh:* PAGER=/usr/bin/less; export PAGER
  *bash:* export PAGER=/usr/bin/less
  *tcsh:* setenv PAGER /usr/bin/less
- Execute an external command (sh)

```
# somecommand
somecommand: command not found
# echo $PATH
/home/tim/bin:/usr/local/bin:/usr/bin:/bin
# pwd
/home/tim/bin/project1
# ./somecommand
Hello world!
# /home/tim/bin/project1/somecommand
Hello world!
# PATH=$PATH:`pwd`; export PATH
# somecommand
Hello world!
```

# UNIX: Bourne SHell script syntax

- The first line of a *sh* script *must* start as follows:
  #!/bin/sh
- Any unquoted # is treated as the beginning of a comment until end-of-line
- Every line is first parsed for shell *metacharacters*. These include characters that the shell will do something with and include:
  # ‘ “ & > < $ % * [ ] ? ! ` ~ ; | , { }
- Distinct commands may be separated by end-of-line, semicolon, or comma
- Environment variables are $EXPANDED
- “Back-tick” subshells are executed and `expanded`
- Pipelines are created | joining the output of | one program | with the next
- Any commands left over must be *builtins* or *external* commands.
- An error will fail the pipeline, but **the script will continue!**

# Unix Pipelines: Pipes are smokin'!

- Pipes take the output of the first program and feed that output into the input of the next program.
- Also sometimes known as "filters".
- Examples:

```
last | less

last | grep ^root | less

last | grep ^root | cut -d -f 2 | less

grep "error" something.out | tail -1
```

# Unix redirection: Lesser and Greater

- >&*filename* redirects the standard output and error to the file called *filename*:

```
last | grep ^root  >& root-logins.txt
less root-logins.txt
```

- >*filename* redirects just standard output
- Don't Clobber me!  By default, > will overwrite existing files, but you can turn this off using shell settings and/or environment variables.
- Appendicitis!  You can append to existing files this way:
  - *sh: >>filename >&1*
  - *csh: >>&filename*
- Use < to redirect a file to a command's *standard input*

```
# cat calculation.txt
(3+2)*8
# bc < calculation.txt
40
```

- Useful when a program does not already query the command line for files to read

# Unix Shell Scripting: Conditional Execution

- program1 && program2
  - Program 2 will execute if and only if program1 exited with a 0 status
  - Example:
    - project1 && echo “Project1 Finished correctly!”
- program1 || program2
  - Program 2 will execute if and only if program1 exited with a *non*-0 status
  - Example:
    - project1 || echo “Project1 FAILED to complete!”
- Exit a script with an error:
    - exit 1

# UNIX commands for programmers

- man –k — Search man pages by topic
- time — How long your program took to run
- date — print out current date/time
- test — Compare values, existence of files, etc
- tee — Replicate output to one or more files
- diff — Report differences between two files
- sdiff — Report differences side-by-side
- wc — Show number of lines, words in a file
- sort — Sort a file line by line
- gzip — Compress a file
- gunzip — Uncompress it
- strings — Print out ASCII strings from a (binary)
- ldd — Show DLLs/SOs program is linked to
- nm — Show detailed info about a binary obj

# Unix Shell scripting: foreach loops

- These are useful when you want to run the same program in sequence with different filenames.
- *sh* example:

```
for VAR in test1 test5 test7b finaltest; do
    runmycode $VAR >$VAR.out
done
```

- *csh* example:

```
foreach VAR ( test1 test5 test7b finaltest )
    runmycode $VAR >$VAR.out
end
```

# Unix job control

- Start a background process:
  - program1 &
  - program1
    *Hit* CTRL-Z
    bg
- Where did it go?
  - jobs
  - ps
- Terminate the job: kill it
  - kill %*jobid*
  - kill *pid*
- Bring it back into the foreground
  - fg %1
- Start a job in the future
  - at

# Regular Expressions

- Powerful language for specifying strings of text to be searched and/or manipulated.
- Used by
  - grep "Get Regular Expression and Print" – search files line by line
  - sed Simple Editing tool, right from the command line
  - awk Scripting language, executes "program" on matching lines
  - perl Pathological Rubbish Lister. Powerful programming language
- Note: These are *not* "file-globs". The syntax is similar, but the semantics are slightly different!
- Cannot be used to match nested structures

# Regular Expressions: Summary

- Fundamentals:

  Match the specified character unless it is a ...

  | | |
  |---|---|
  | . | Match *any* character (except EOL) |
  | [*character class*] | Match the characters in *character class.* |
  | [*start-end*] | *start* to *end* |
  | [^character class] | Match anything *except* the character class. |
  | $ | Match the end of the line |
  | ^ | Match the beginning of the line |
  | * | Match the preceeding expression zero or more times |
  | ? | Match the preceeding zero or one time |
  | \| | Match the lef hand side OR the right side |
  | (regexp) | Group the regular expression |
  | \ | Treat next character literally (not specially) |

- Examples:

  Match a line beginning with a space-padded line number and colon.

  ```
  ^[ \t]*[0-9][0-9]*:
  ```

  Match my name (various spellings)

  ```
  (Tim Shelling)|(TJS)|(T\. Shelling)|(Timothy J\. Shelling)
  ```

  Match if the line ends in a vowel or a number:

  ```
  [0-9aeiou]$
  ```

  Match if the line begins with anything but a vowel or a number:

  ```
  ^[^0-9aeiou]
  ```

# Getting files from and to Unix

| | |
|---|---|
| Windows PC | SAMBA<br>FTP/SFTP |
| DOS/Win Floppy Disk | mcopy, mdir, mdel, etc |
| Internet | FTP, ncftp<br>lwp-download<br>mail |
| Archives | ar<br>tar<br>zip, unzip *(if available)* |