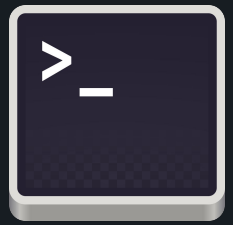


Command Line Fundamentals

Day 2

Follow Along At sipb.mit.edu/iap/2022/clf/

Executing Commands as Super User

- Like in Windows, you may need to run certain programs and commands with elevated privileges

- In Unix, this concept is represented as the “super user”. There are two main ways to become the super user:
  - su - Login as the ‘root’ user (Note the dash!)
  - sudo *command* - Execute *command* with superuser privilege. Requires current user to be in the sudo group.

- On some systems, the root user is disabled. In that case, you will need to specify the username of an admin account, like su - admin1

Security Concerns with Sudo

- Least Privilege Principle: Users and programs should have the lowest privilege/permission level necessary to do their tasks
  - Else, it can be dangerous when executing irreversible actions

- Sudo allows commands/programs to access entire system (e.g. remove/add users, install new software, view user-specific files, etc.)
  - Ex: sudo rm -rf /

- Rule of Thumb:
  - Avoid running sudo where possible! Often times you don’t need sudo to run a command

Adding and Managing Packages

- On personal machines, you will often need to install external packages

- Different OSs use different package managers.
  - Ubuntu has apt
  - Most MacOS users use homebrew (Note, Homebrew is not installed by default)
  - Others include yum, flatpak, pacman, etc.

- Package managers make it easy to update and modify your packages
  - Most of updates/modifications will use root privileges

Ubuntu - Apt

- For installing/uninstalling:
  - sudo apt install *package* (Note: apt is generally preferred over apt-get)
  - sudo apt remove *package* (Removes package from system)

- For updating:
  - sudo apt update (Updates the system's package repository)
  - sudo apt upgrade (Actually updates software)

MacOS - Homebrew

- Homebrew is not installed by default. Visit brew.sh to run their one-line installation.
- For installing/uninstalling:
  - brew install *package* (Note: Does not require sudo for installs)
  - brew uninstall *package*
- For updating:
  - brew update (Update all package *definitions*)
  - brew upgrade (Upgrade everything)
- If you want to install programs using a graphical interface, look into Homebrew Cask

Examples

Ubuntu

```
# Install vlc
sudo apt install vlc -y

# Install Node.js v14.x
curl -fsSL https://deb.nodesource.com/setup_14.x | sudo -E bash -
#Note, don't normally do this! Curling into sudo is a bad idea
sudo apt-get install -y nodejs
node --version
```

Python Libraries

- Similar to OS package managers, Python has pip as its own package manager
- Installation:
  - pip install *package*
- Uninstallation:
  - pip uninstall *package*
- If you have both Python 2 and Python 3 installed on your system, you will need to use pip2 or pip3 instead to install Python 2 or Python 3 packages, respectively.
- If you're worried about mixing library versions, consider using *virtual environments*!

Bash Scripting

Shells include scripting languages with their own variables, control flow, and syntax. Useful for automating tasks. Today's focus is on bash.

Defining a variable (Note, do not add spaces before/after the "=")

```
var1=’a’

var2=5
```

Access value with $var_name

Print values with echo $var_name

Note, ' defines literal strings, where “ will attempt to substitute in variables

Bash Scripting - Loops and Conditionals

For loops

```
#!/bin/bash
for value in $(seq 1 5);
do
   mkdir $value;
done

search_dir=. #current directory
for entry in "$search_dir"/*
do
   echo "$entry"
done
```

If/else conditions

```
#!/bin/bash
# If the user is root, its UID
is zero.
if [ $UID  -eq  0 ]
then
 echo "You are root!"
else
 echo "You are not root."
fi
```

Command Aliases

- Aliases allow you to provide a shorter name for frequently used commands.
- By default, Ubuntu has many built-in aliases. View with command alias
- To add permanent aliases, edit your ~/.bashrc (Ubuntu) or ~/.bash_profile (MacOS) file
- Example:
  - alias lsa='ls -a' (List hidden files)
  - alias rmi='rm -i' (Prompt you before every removal)
  - alias ssh-mit='ssh tim@athena.dialup.mit.edu' (Shorten ssh command)

Searching

- Use grep to search within files (can also search multiple files at once)
  - Useful for finding where function is defined in large codebase
- Syntax: grep *pattern file(s)*
  - Example: grep quarry hamlet.txt (returns *all* lines of *Hamlet* with word “quarry”)
- Tip: Use an asterisk (*) for file matching
  - Ex: grep *pattern* * to search pattern within all files (not subdirectories) of current directory
- Can search folders recursively with -r flag (e.g. grep -r *pattern*)

- Use find to (recursively) find specific files in a directory. Must be accompanied with a flag, with the most useful one being -name
  - Ex: find -name example.txt

Extracting Info From Files

- sed is a stream editor. In other words, it takes in a file as input and prints out some output using that file's contents (but we aren't modifying the file)
  - To modify the file, use the flag -i
  - Involves knowing Regex
  - Basic use case:
    - sed "s/a/A/" file (Replaces a with A, in the first instance of each line)
    - sed "s/a/A/g" file (Replaces a with A, in all instances)
    - Concatenate sed commands using a semicolon (;)
      - sed "s/a/A/g;s/b/B/g" file (Turns all a and b into uppercase)

Extracting Info From Files (cont.)

- awk is used for pattern scanning and processing. Searches a file to find specified patterns and then performs an action on those patterns.
  - Doesn't modify original file
  - $n: Represents the nth "column"
  - Basic use case: awk '{print $1, $2}' awk-ex.txt

Controlling Your Output

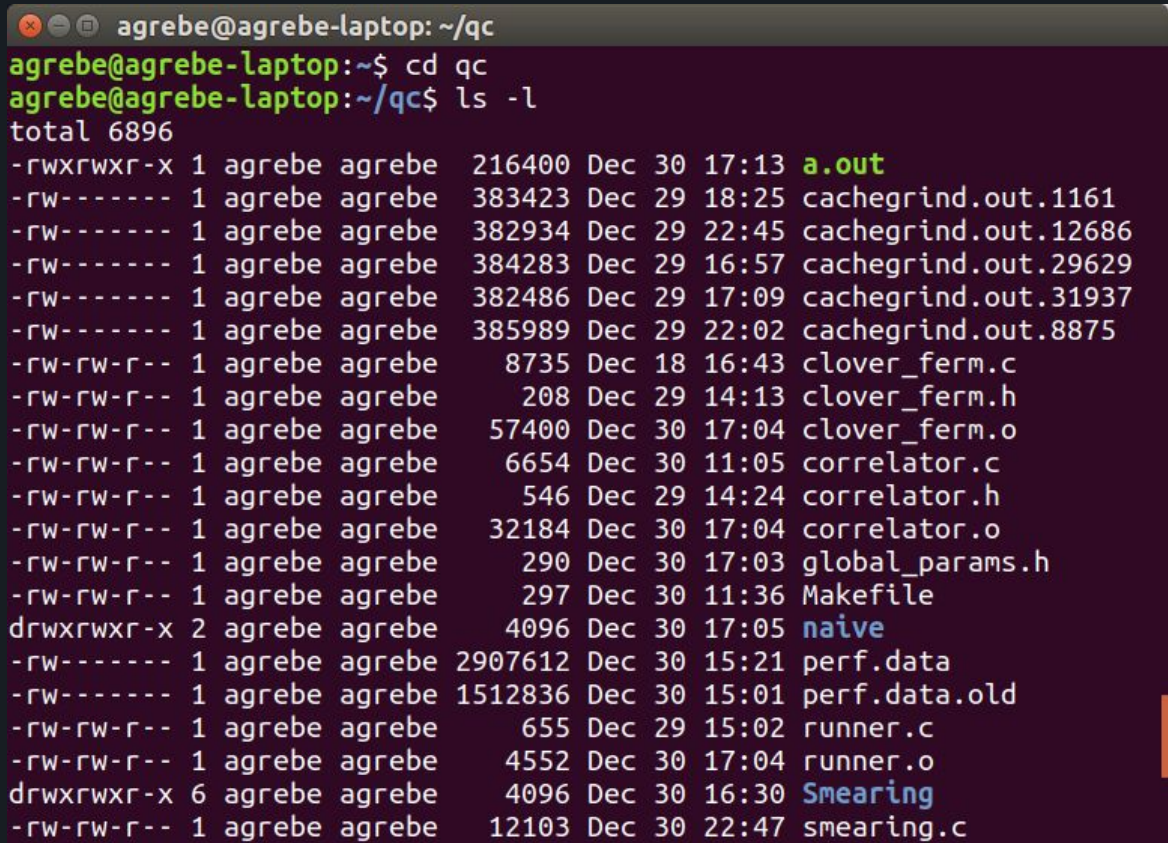
- Piping
  - Use of | to send output of command to another command
  - Ex: who | awk '{print $1}' (Prints only the usernames of people in server right now)
  - Note, you can daisy chain with piping
- Redirecting
  - Use of > to send output of command to a file
  - Ex: history > history.txt (Saves your history onto a file named history.txt)

Helpful Keybindings

- TAB - "Auto-completes" the remainder of a file directory or command
  - If multiple exist with the same beginning, the beginning part is filled in, and you need to add more specific characters to tab-complete again
- CTRL + a - Move cursor to beginning of line
- CTRL + e - Move cursor to end of line
- CTRL + w - Delete word left of cursor
- CTRL + u - Clear text from beginning of line up to cursor
- CTRL + k - Clear text from end of line up to cursor

File Permissions

- Permission bits of ls -l give info about file type and ownership
  - d = directory
  - r = readable
  - w = writable
  - x = executable
- 1st bit specifies file type
- Then, 3 sets of permission bits (owner, group, and global)
  - First 3 bits are for file owner permissions
  - Next 3 bits are for group permissions
  - Last 3 bits are for global permissions



Changing File Permissions

- chmod affects file permissions
- Can be added with + and removed with -
  - chmod +x *file* makes *file* into an executable for all
- Group (g), other (o), and all (a) can also be set
  - chmod a+rx makes file globally readable and executable
  - chmod a+w gives all users write access – **dangerous on shared servers**
- Note: On most machines, system admin (root) has universal read/write/execute access by default

Transferring and Compressing Files

- Secure copy (scp) used to transfer files to and from servers
  - scp *file server:/destination/on/server* (Copy file from local machine to server)
  - scp *server:/file/path/on/server destination* (Copy file from server to local machine)
- Syntax is similar to cp but includes server name
  - Use -r for copying directories
- When you have many files, recommended to zip/tarball them first
  - tar -czf archive.tar.gz dir/* (Creating a tarball named archive with all files in dir)
  - scp *server:*archive.tar.gz . (Copying them into our local machine)
  - tar -xzf archive.tar.gz (Unpacking the tarball)
- MIT note: Ethernet (100 MB/sec) is much faster than WiFi (5 MB/sec)

Background Jobs

- While job is running, CTRL + z pauses it, and bg sends it to background
  - Job continues running (and displaying output)
  - You regain input prompt in the same terminal session
- Can also start process in background by appending command with &
  - Ex: sleep 10 &
- You can run multiple jobs simultaneously
  - for i in `seq 1 10`; do sleep 10 & done
- CAUTION: Be careful not to overwhelm/crash system with processes
  - Particular danger: Fork bomb (where each process spawns two others)
- wait command stalls until all background jobs completed

Termination of Background Jobs

- All jobs finish when script terminates or terminal exits
- jobs lists currently running/stopped jobs (typically backgrounded)
- Scripts: Insert wait at end of script to prevent premature termination
- To survive end of terminal, preface job with nohup
- nohup will redirect job output to file (default: nohup.out)
  - Job no longer confused by lack of terminal to print to
- nohup also makes job persistent after terminal exit
  - Useful for long-running non-interactive jobs

Persistent Terminal Sessions

- tmux (“terminal multiplexer”) provides interactive terminal session that persists across logons
- Can be used similar to normal terminal and then detached with CTRL + b, d
  - Note: Release CTRL before typing d
- Reattach window with tmux a
- Can also have multiple windows that can be attached/detached
- Note: With MIT dialup servers, need to log onto same server machine from previous session
  - athena.dialup.mit.edu redirects to one of several server machines (scrubbing-bubbles, ten-thousand-dollar-bill, etc.) to balance load
  - Unlike most files, tmux session associated with just one machine
- screen has similar functionality (but overrides useful keybinding CTRL + a)

Graphics

- Graphics on Linux handled by X Window System
  - Developed at MIT for Project Athena!

- On Linux, can enable graphics forwarding over SSH with -X flag
  - ssh -X athena.dialup.mit.edu

- Note: Sometimes finicky, almost always slower than command line

Thanks for coming!

Slides can be found at sipb.mit.edu/iap/2022/clf/

Feel free to contact any three of us if you have questions:

Anthony Grebe agrebe@mit.edu

Javier Solis javsolis@mit.edu

Huy Dai huydai@mit.edu

Special thanks to Ashay, CJ, and other SIPB folks for helping us set up these lectures