CSCI397: Tools for the Software Life Cycle and Beyond

Software Tools
- What are they?
  - What is the goal of software tools?
- Why do we use them?
- Why do we develop them?

Goal: Productivity
- Many available tools
  - UNIX & UNIX-like systems (e.g., Linux)
  - Open-source (Gnu, Apache, Eclipse)
  - Proprietary
  - Variety of purposes
- Know what (mostly, free) tools are available, what they do, how to use them

Goal: Automation
- Often have to do a task over and over again
  - Time-intensive to do by hand
  - Shortcuts aren’t enough
- What we want
  - Tools to make tasks easier
  - Scripts to be able to repeat the tasks easier
Main Types of Tools

- Command-line
- Graphical/GUI interfaces

What are the benefits and limitations of each type of tool?

Command-Line Tools

- Benefits
  - Flexible—lots of options
  - After run once, can run again in same terminal using up arrow key or using !command
  - Tab-completion
  - Automation: Can be put into bash scripts and repeated

- Limitations
  - Requires knowing name of command
  - Requires knowing syntax of command, options
    - Easy to screw up!
  - Slower learning curve

GUIs

- Benefits
  - Require less knowledge of syntax
  - Generally: faster learning curve

- Limitations
  - Can require many clicks to do even simple operations
  - May require a lot of set up/configuration
  - Harder to automate, repeat tasks

Course Content

- Unix tools
- Bash scripting
- Software development tools
Software Life Cycle: Waterfall Model

Our Focus

Iterative Design

Requirements

Design

Test/
Get feedback from
users

Evaluate

Implement

Deployment

Course Objectives

• At the end of this course, you will be able to
  ➢ Use a variety of Unix tools
  ➢ Apply a variety of tools to automate many tasks
  ➢ Describe the use of state-of-the-art software tools for developing and maintaining large software systems, based on hands-on experience
  ➢ Discuss when best to use different tools, the limitations of the tools, and what they have to offer
  ➢ Discuss the challenges and strategies in building software tools
  ➢ Communicate technical content in both oral and written forms

Non-Syllabus Goals

• Improve your productivity
• Unix confidence/proficiency
  ➢ To intermediate user
• Tool confidence
  ➢ Less intimidated by installing, learning new tools
• Resume builder!
  ➢ Impress potential employers, advisors
• Non-goal: System Administrator


Expectations

• Material is most relevant in context
  ➢ Need to make it relevant to you
  ➢ What would you like to do—now or in the future?
  ➢ What tools interest you?

• Actively explore tools
  ➢ Try out everything we do
  ➢ Make mistakes and learn from them

Grading

• (47%) Individual programming, reading, and homework assignments
• (15%) Quizzes
• (30%) Tool Demonstrations
• (8%) Professionalism: participation and attendance

Our Heroes: UNIX Developers

UNIX

Ken Thompson
Dennis Ritchie
UNIX Philosophy

- Doug McIlroy, inventor of Unix pipes, a founder of Unix tradition:

  *This is the Unix philosophy:*
  
  Write programs that do one thing and do it well.
  Write programs to work together.
  Write programs to handle text streams, because that is a universal interface

- This is usually severely abridged to “do one thing and do it well”

---

UNIX Philosophy

- Make each program do one thing well
  
  Relation to software design principles?

---

UNIX Philosophy

- Make each program do one thing well
  
  - More complex functionality by combining programs
  - Make every program a filter
  - More efficient
  - Better for reuse

---

The UNIX Philosophy

- Scripting increases leverage and portability

  List the usernames of a system’s current users:

  ```
  who | awk '{print $1}' | sort | uniq
  ```

  We’ll talk more about piping on Wednesday...
The UNIX Philosophy

- Avoid captive interfaces
  - The user of a program isn’t always human
  - Look nice, but code is big and ugly
  - Problems with scale
- Silence is golden
  - Only report if something is wrong
- Think hierarchically

UNIX Highlights / Contributions

- Portability
  - Because implemented in C rather than assembly language (specific to machine), ran on variety of machines
- TCP/IP implementation -- 1984
  - Communicate btw different machines from different vendors
- Hierarchical file system; the file abstraction
- Multitasking and multiuser capability for minicomputer

UNIX Highlights / Contributions

- Inter-process communication
  - Pipes: output of one programmed fed into input of another
- Software tools
- Development tools
- Scripting languages

Quotes

- "Unix is simple. It just takes a genius to understand its simplicity." – Dennis Ritchie
- "UNIX was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things." – Doug Gwyn
- "Unix never says 'please'." – Rob Pike
- "Unix is user-friendly. It just isn't promiscuous about which users it's friendly with." – Steven King
- "Those who don't understand UNIX are condemned to reinvent it, poorly." – Henry Spencer
The Operating System

- The government of your computer
- Kernel: Performs critical system functions and interacts with the hardware
  - Loaded into memory during the boot process, and always stays in physical memory
  - Responsible for managing CPU and memory for processes, managing file systems, and interacting with devices
- Systems utilities: Programs and libraries that provide various functions through system calls to the kernel

UNIX Structural Layout

- Kernel: lowest-level, or ‘inner-most’ component
- User Space
  - shell scripts
  - system calls
  - utilities
    - C programs
    - compilers
- Devices
  - terminal
  - disk
  - device drivers
  - scheduler
  - swapper
- Devices
  - printer
  - RAM

UNIX System Structure

- User
  - c programs
    - scripts
  - shell and utilities
    - ls
    - ksh
    - gcc
    - find
  - kernel
    - open()
    - fork()
    - exec()
  - hardware
What is a Shell?

- User interface to the operating system
- A program like any other
- Command-line interpreter
  - Functionality:
    - Execute other programs
    - Manage files
    - Manage processes
- Basic form of shell:
  ```
  while <read command>:
    parse command
    execute command
  ```

Most Commonly Used Shells

- `/bin/sh` The Bourne Shell / POSIX shell
- `/bin/csh` C shell
- `/bin/tcsh` Enhanced C Shell
- `/bin/ksh` Korn shell
- `/bin/bash` Free ksh clone

Which shell do we use in the lab?

Shell Interactive Use

- When you open a terminal, you interactively use the shell:
  - Command history
  - Command line editing
  - File expansion (tab completion)
  - Command expansion
  - Key bindings
  - Job control

Shell Scripting

- A set of shell commands that constitute an executable program
- A shell script is a regular text file that contains shell or UNIX commands
- Very useful for automating repetitive tasks and administrative tools and for storing commands for later execution

More on this later…
Simple Commands

- Sequence of non-blank arguments separated by blanks or tabs
- 1st argument (numbered 0) usually specifies the name of the command to be executed
- Any remaining arguments:
  - Are passed as arguments to that command
  - Depending on command, arguments may be filenames, pathnames, directories or special options
  - Special characters are interpreted by shell

Example of Simple Command

```
$ ls -l /bin
-rwxr-xr-x  3 root   root   63216 Sep 7  2006 zcat
$ prompt command arguments
```

- Execute a basic command
- Parsing into command and arguments is called splitting

Types of Arguments

```
$ tar --c --v --f archive.tar main.c main.h
```

- Options/Flags
  - Convention: -X or --longname
- Parameters
  - May be files, may be strings
  - Depends on command

TODO

- Check out the course web site