Interactive shells

- Most actual operating system commands highly specialized, not ideally suited for direct interaction with user
- Most systems provide a variety of command interfaces (shells) to interact with user
- When you are tapping or clicking on icons, or typing commands in a command window, such a shell interprets your actions and invokes appropriate programs or OS commands to handle them

Many many shells out there

- Many different shells supported on different operating systems, offering different sets of features/support for the user
- Some popular shells supported on many different operating systems
- The default shell used on our csci servers is Bash, we'll discuss it at length this term
- Whenever you type a command, the bash interpretter reads it and decides how to handle it and give you feedback

Complex tasks and shells

- As developer, we tend to carry out many tasks that involve long sequences of (sometimes complex) shell commands, often repeating these task daily, weekly, monthly, etc, e.g.
  - Performing an install of complex software
  - Configuring a new machine
  - Carrying out a backup and generating a status report
  - Setting up a test environment for a program, running a suite of tests, and generating a report on the results

Need for programmatic support

- Complexity of commands + need to do them in sequence means we should store them and somehow tell the shell “do these”
- Many of the commands involve choices, e.g. “if there is enough disk space do X, otherwise do Y”
- Many of the commands involve getting file and directory names, or other arguments, from other programs, files, or the user
- Scripting languages are programming languages to allow user to write programs (scripts) to interact with the shell, many such languages
- Allows task automation: improving efficiency, reducing errors

Bash

- We can write scripts (command collections) in bash, and when we run them the bash interpretter will carry out the specified commands
- Note that languages like bash are highly text-based – they're meant to handle commands typed by the user, and to deal extensively with files, directories, and system commands
- This specialization results in some unusal syntax and behaviour compared to languages like C++, Java, etc

Creating an executable script

- Put our bash commands into a file, usually given the .sh extension to show it is a script (remember the #! line)
- From the command line, we make the script executable, e.g. “chmod u+x mystuff.sh”
- We run the file (and hence all the commands in it), e.g. using ./mystuff.sh