Lecture A for foreign students

Lecture 2: Command line

Norbert Pozar (npozar@se.kanazawa-u.ac.jp)

April 26, 2018

Today’s plan

- Using the command line efficiently

Using the command line efficiently

Basic commands

```
$ command arg1 arg2 ...
```

`ls`	list files
`pwd`	show current directory
`cd`	change directory
`which`	location of command

Basic commands

`Up`	previous command
`Ctrl-U`	erase the line
`Ctrl-R`	search previous commands
`Tab`	completion

Getting help

Most Unix commands provide help:

```
$ command --help
$ man command
```

Standard IO

Commands communicate with other programs via standard input (**stdin**) and output (**stdout**, **stderr**).

- input and output streams of text.

C/C++'s

`printf` and `cout << ..` print to the standard output.

`cin >> ..` reads from the standard input.

Python

`print`

Working with standard IO

`command > file` Redirect standard output of `command` into a file `file` (overwrites previous content).

`command >> file` As `>`, but *append* to the file (keeps previous content).

`command < file` Load `file` and feed it into `command`'s standard input.

`command1 | command2` "Pipe"the standard output of `command1` into the standard input of `command2`.

Using stdin for inputs

read_stdin.cpp

```cpp
#include <iostream>

using namespace std;

int main() {
    int i;
    double x;
    cout << "Enter two numbers" << endl;
    cin >> i >> x;
    cout << "You entered: " << i
         << " and " << x << endl;
    return 0;
}
```

Using stdin for inputs

```
$ g++ read_stdin.cpp -o read_stdin
$ ./read_stdin
```

Using stdin for inputs

```
$ g++ read_stdin.cpp -o read_stdin
$ ./read_stdin
```

```
$ echo 1 2 | ./read_stdin
```

or save data in a file first:

```
$ echo 1 2 > nums.dat
$ ./read_stdin < nums.dat
```

Useful commands

`cat file`	Read `file` and print it to stdout.
`echo args...`	Print `args...` to stdout.
`head`, `tail`	Read stdin and print the first (last) 10 lines to stdout.
`grep pattern`	Read stdin and print lines that contain pattern.
`less`	Read stdin and show it one page at a time.
`wc -l`	Read stdin and print the number of lines read.

C/C++ program `main` return value

What does the returned `int` represent?

```
int main() {
    return 0;
}
```

Exit code

Exit code is a numerical value that every process returns.

It is the `int` return value in `int main()`:

0	Everything went OK.
nonzero	Indicates an error.

Using the exit code

`$?` Shell variable that contains the last executed command's exit code:

```
$ echo $?
```

`cmd1 && cmd2` Run `cmd1` first, and if it finishes successfully (exit code is 0), run `cmd2`.

`cmd1 || cmd2` Run `cmd1` first, and if `cmd1` fails (exit code is not 0) run `cmd2`.

Fast build-run cycle

Build and run immediately if the build succeeds:

```
$ gcc main.c && ./a.out
```

Using command line arguments in your C/C++ program

Command line arguments are passed into `main` function:

```c
// args.c
#include <stdio.h>

int main(int argc, char *argv[])
{
    for (int i = 0; i < argc; i++) {
        printf("argv[%d] = %s\n", i, argv[i]);
    }
}
```

Running computations with different parameters

Use

- command line arguments
- stdin
- configuration files (simple text, JSON, …)

to set parameters that you often need to change.

This allows for scripts to run many simulations automatically and reproducibly.

Do not

- Edit your source file to change such parameters.

This is slow, **tedious** and very error prone.