Command Line Interface:
Writing Programs on a Remote Computer

Debra Deppeler

December 9, 2015 for cs302

Most of us greatly appreciate the convenience and time-saving features of an Integrated Development Environment (IDE) like Eclipse. However, there are times more simple programing development tools will suffice. Command line interfaces (CLIs) have been around since long before Graphical User Interfaces (GUIs). They are fast, fun, and often more flexible if not also more functional. Typically, the user has control over many more environment variables than the GUI version supports.

How to Connect to a Remote Computer UW-Madison Computer Sciences

Launch a terminal Emulator on your local machine

Windows

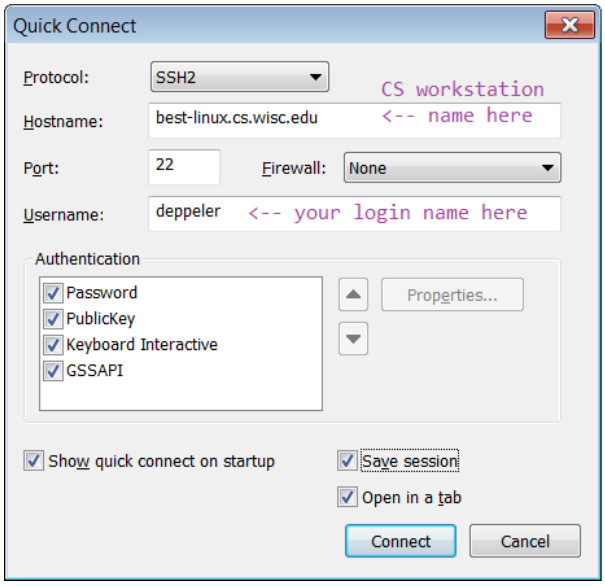
1. Download Putty or SecureCRT (search the DoIT shelf)
2. Install Putty or SecureCRT (follow installation instructions)
3. Launch (open) Putty or SecureCRT

MacOSX

1. Find and launch the terminal application

What do I do once my terminal application is running?

1. If you have a CS account, use Quick Connect or connect using `ssh`:
   `% ssh best-linux.cs.wisc.edu`



2. Enter your CS login name (if not already entered) and your CS password and press Enter.

What's next?: Basic Command-line Programming Tools

Learn Some Basic Linux Commands

pwd print (display) working directory

ls list file names of files and directories in working directory

mkdir *dirname* create a new subdirectory in the current working directory

cd *dirname* change to the directory

Learn Some Command-Line Programming Tools

pico a text-only file editor (good for beginners)
– type **Ctrl-x** to exit and follow prompts to save

nano a text-only file editor (also good for beginners)
– type **Ctrl-x** to exit and follow prompts to save

emacs a text-only file editor (some programmer's favorite)
– type **Ctrl-x Ctrl-c** to exit and save buffers

vi a text-only file editor (has a bit of a learning curve, but Deb's favorite)
– type **Shift-ZZ** to write (save) file and exit

javac compile a Java program to bytecode (.class file)

java run a Java program (bytecode)

Example Shell Session

The following is an example of a shell session in which a Java program is created (edited), translated (compiled), and executed (run).

```
================================================================================
  This instructional workstation is installed with RedHat Enterprise Linux 6

 If you encounter any issues, please send a problem report to lab@cs.wisc.edu
================================================================================
      REMINDER: NO FOOD or DRINK IN THE CS INSTRUCTIONAL COMPUTER LABS
            NEVER POWER DOWN WORKSTATIONS IN THE COMPUTER LABS
================================================================================
galapagos-14[~]% pwd
/afs/cs.wisc.edu/u/d/e/deppeler
galapagos-14[~]% ls
private
public
galapagos-14[~]% mkdir programs
galapagos-14[~]% mkdir programs
mkdir: cannot create directory 'programs': File exists
galapagos-14[~]% mkdir programs/JavaDemo
galapagos-14[~]% cd programs/JavaDemo
galapagos-14[~]% pico JavaDemo.java

                    ... pico editor window opens and user may type source code
```

Edit

Note: `pico` is a file editor and will open a file for writing. Type the source code for a Java program. Be sure to include a *main* method if you wish to be able to run (execute) the program as a stand-alone program.

```
public class JavaDemo {
    public static void main(String []args) {
        if ( args.length > 0 )
            System.out.println("Hello " + args[0]);
        else
            System.out.println("Hello World!");
    }
}
```

Type **Ctrl-x** to save your file and exit the `pico` editor. Follow prompts to (overwrite) file or save with a different file name, and exit program. Once the program source code has been written and saved to a java file, you can compile and run the program.

Compile

Use `javac sourcecodefilename.java` to compile (translate) the source code to *bytecode*. If there are no compiler errors, and there is a *main* method defined in the class, you can run the Java program.

Run

Use `java bytecodefilename` to run the bytecode with the Java Runtime Environment (JRE).

```
galapagos-14[~/programs/JavaDemo]\% javac JavaDemo.java
```

```
galapagos-14[~/programs/JavaDemo]\% java JavaDemo.class
Error: Could not find or load main class JavaDemo.class
galapagos-14[~/programs/JavaDemo]\% java JavaDemo
Hello World!
galapagos-14[~/programs/JavaDemo]\% java JavaDemo.
Error: Could not find or load main class JavaDemo.
galapagos-14[~/programs/JavaDemo]\% java JavaDemo student
Hello student
galapagos-14[~/programs/JavaDemo]\% ls
JavaDemo.class  JavaDemo.java
galapagos-14[~/programs/JavaDemo]\% ls -al
total 8
drwxr-x---  2 deppeler deppeler 2048 Apr 27 09:53 .
drwxr-x--- 42 deppeler deppeler 4096 Apr 27 09:50 ..
-rw-r-----  1 deppeler deppeler  648 Apr 27 09:53 JavaDemo.class
-rw-r-----  1 deppeler deppeler  210 Apr 27 09:53 JavaDemo.java
galapagos-14[~/programs/JavaDemo]\%
```

What else can I do in linux shell?

javadoc *.java run javadoc utility to build javadoc web pages for all java files in current directory

cp source.txt destination.txt copy source.txt to new file with name destination.txt

ls -al list acls (permissions) and other file info for files and directories in working directory

ls *dirname* list file names of files and directories in named directory

grep *text files* find files that contain *text*

fs la . list the file acls (permissions) for this directory

fs sa . system:anyuser read set current directory permissions so anyone can read this current directory

afs_rseta . system:anyuser none recursively set directory permissions so other users can not read this directory or those below (be sure you still can)

rmdir *dirname* remove an existing directory

man *command* read the built-in manual for *command*

Makefile: a simple way to automate your build (or test) process

A `Makefile` (case-sensitive) is a text-only file that contains build information for building program executables. For Java, this means it compiles the source code, and then runs the program using the java (JRE) tool. Use your favorite editor to create a document named `Makefile` and type the following as the contents of the file.

```
make:
    javac *.java
    java MainClassName
```

Exit and save the file and then type `make` at the terminal's command prompt. The java files will be compiled and if there are no compiler errors, the program in `MainClass.java` will be run.
Note: The lines that follow a *make* label must have a tab character as their first character.
Add a section labeled `clean` to your `Makefile` to give yourself an easy way remove `.class` files and get a clean build. Execute this section with the command `make clean`. The first section is the section that will be run if make is run without any label, `make`.

```
make:
    javac *.java
    java MainClassName

test:
    javac *.java
    java TestClassName testArg1 testArg2 testArg3 ...

clean:
    \rm *.class
```