Lund University
Computer Science
Christoff Bürger, Görel Hedin, and Niklas Fors

Compilers
EDAN65
2016-08-28

Programming Assignment 0
Java and the Unix command line

It might be some time since you used Java and the Unix command line, but in this course you need to be fairly proficient with these tools. Therefore we provide this voluntary assignment as a refresher.

You can do this assignment either on the student lab machines, or on any other Unix computer, e.g., Mac or Linux. Skip any section that practices things you are already familiar with.

There are student lab machines with Unix in the following rooms:
`venus, mars, jupiter` (north part of E-house basement)
`hacke, panter, lo, val, falk, varg` (south part of E-house basement)
`alfa, beta, gamma` (2nd floor, E-house)
`backus` (entry level, Math house)

1 The Unix file system

Can you do the following tasks without looking in a manual?

- Open a terminal window.
- Go to your home directory. What is its full path name? How many files are there?
- Get help for a command, by looking in the manual. For example, can you find out what the `-t` option does for the `ls` command?
- Go to the parent directory. What files are there and which access rights do they have?
- Go to the `/bin` directory. Do you recognize any of the files there?
- Go to your home directory.
- Create a new directory `my-dir`.
- Create a new file `my-file` (you can use the `touch` command to create new empty files).
- Copy `my-file` to `my-dir`; rename it to `my-file.txt` in the process.
- Delete `my-file`.
- How could you speed up the last two steps? Use the move command to undo them.
- Create a new directory `my-dir-2`. Move `my-file` to it.
- Copy `my-dir-2` to `my-dir`.
- Imagine you would like to move `my-dir-2` instead of copying it. Delete the copy `my-dir/my-dir-2` and move `my-dir-2` to `my-dir`. Use `ls` if you get confused what is where.
- Delete all stuff you created throughout the tasks (should be just the directory `my-dir` and its contents).

If you are sure you can do all these tasks without looking in a manual, you can skip the rest of this section and go to Section 2. If not, go through the rest of this section, then go back and do the tasks above.

If you don't know how to open a terminal window on your platform, google this. For example, google

```
open terminal window in ubuntu
```

if you are running on the Linux Ubuntu platform.

1.1 Navigation

Play around with the following commands to understand how you can navigate in the Unix file system: *Note!* If you don't know how to type the tilde character (~) on your platform, google this.

pwd (print working directory)

cd (change directory to your home directory)

cd dir (go to the *local* directory `dir`, relative to the current directory)

cd /usr/bin (go to the directory with the *absolute* path `/usr/bin`, not relative to the current directory)

cd .. (go to the parent directory)

cd ~/foo (go to the directory `foo` in your home directory)

ls (list all files in the working directory; files starting with a period like `.gitignore` are excluded)

ls -a (list all files in the working directory, including those starting with a period)

ls -l -h (list files with detailed information like size, change time and access rights)

man ls (Show the manual for the ls command. Type space to go to the next page. Type `b` to go to the previous page. Type `q` to quit.)

1.2 Manipulation

Play around with the following commands to understand how you can manipulate files and directories. *Note!* There is no undo in Unix! **Be careful** not to delete or overwrite important files.

mkdir d (make directory `d`)

cp dir/f1.mk f2.txt (copy file `dir/f1.mk` and name the copy `f2.txt`; the copy will be created in the working directory, the original is in subdirectory `dir`)

cp f .. (copy file `f` to its parent directory)

cp ../f . (copy file `f` in the parent directory to the working directory.

cp -r d1 d2 (recursively copy directory `d1` into directory `d2`; copies `d1` and all its content)

rm f (remove file `f`)

rm -r d (recursively remove directory `d`; deletes `d` and all its content)

mv x y (assume `x` is a file and `y` an directory: move `x` to `y`)

mv x y (assume `x` is a file and `y` does not exist: rename `x` to `y`)

mv x y (assume `x` and `y` are files: overwrite `y` with `x` and remove `x`)

mv x y (assume `x` is a directory: rename it to `y` if `y` does not exist yet, otherwise move it to `y`; the operation is aborted with an error if `y` is an already existing file)

1.3 Further reading

If you need further help in understanding the commands, you can read, for example:

`http://linuxcommand.org` `->` Learning the shell `->` Navigation/Looking Around/Manipulating Files

Introduktion till LTH:s Unixdatorer, by Per Foreby, 2012. In Swedish.
`http://www.student.lth.se/fileadmin/ddg/text/unix-x.pdf`

1.4 Check your proficency

Now check your proficiency with Unix commands by going back to the beginning of section 1 and do the tasks there.

2 Editing text files

You should to be able to use an ordinary text editor to edit text files. If you are not used to a particular text editor, we recommend Nano. Nano is simple and runs directly in the terminal window. Here is how you get started with it:

nano f (type this in a terminal window to start nano on the file f)

type text (to add text to the file)

use the arrow keys (to navigate in the file)

ctrl-X (to exit the nano editor) (**ctrl-X** means hold down the ctrl-key while you type X.)

ctrl-G (to get a help text. Then type **ctrl-X** to get back to the file you were editing.)

2.1 Create a small Java program by using a text editor

Use the text editor to construct a java program `Hello.java`:

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello!");
    }
}
```

Make sure the file contains the correct code by typing

```
less Hello.java
```

in the terminal window. The `less` command lists the contents of a file, and you can move forward and backward using space and `b`, just like in the `man` command. You can also use the arrow keys. Type `q` to quit from the `less` command.

3 Java from the command line

You need to be able to compile and run Java programs from the command line.

Can you do the following tasks without looking in a manual?

- Implement two Java classes, `A` and `B` in a package `p`, where `B` contains a main method and imports `A`. The main method prints `Hello!` on the standard output and the `toString()` value of an `A` object.
- Compile all classes.
- Run the main method of `B`.
- Pack the classes into an executable Java Archive called `hello.jar` (you have to create a manifest).
- Execute `hello.jar`.

- Add a `while (true) { System.out.println("Foo"); }` loop to the main method of `B`; recompile it. If you execute it, how can you abort?
- Add a string argument to `B` so that it prints `Hello` *arg* `!` instead of only `Hello!`, where *arg* is the argument you give on the command line when you run `B`.

Hints: Classes must be in directories named like their package; their first code line is the respective package name. To compile and run them, your working directory has to be the parent directory of the package hierarchy. Example code to print the number of command line arguments and the first command: `System.out.println{ args.length + ": " + args[0] };`.

If you are sure you can do all these tasks without looking in a manual, you can skip the rest of this section. Otherwise, read through the rest of this section and then go back and do the tasks above.

3.1 Compiling and running Java programs

Here are some commands for compiling and running Java programs from the command line.

javac C1.java C2.java (compile classes `C1` and `C2` to bytecode for the Java Virtual Machine)

javac p/*.java (compile all classes in the package `p`)

javac -d dir p/C.java (compile class `p.C`; generate its class file in directory `dir`)

javac -cp d1:d2 C.java (compile class `C`; `d1` and `d2` are added to the classpath for compilation (the working directory is always included), such that `C` can use classes in `d1` and `d2`)

java p.C (execute the main method of class `C` in package `p`)

java p.C just three args (execute `p.C` with arguments `just`, `three` and `args`)

java -cp d1:d2 p.C (execute `p.C`; `d1` and `d2` are added to the classpath for execution (the working directory is always contained), such that classes in `d1` and `d2` can be used)

java -jar my-jar.jar (execute the main method specified in the manifest of the Java Archive `my-jar.jar`)

`ctrl-c` **while program runs** (abort execution of running program)

3.2 Jar files and classpaths

Java programs are packaged and distributed as JAR (**J**ava **Ar**chive) files. We will now create a JAR file that will contain the `A` and `B` classes defined before. A JAR file requires a Manifest file that specifies, for example, which class is the main class of the JAR file. We will start creating the Manifest file.

Add the following code to the file `MANIFEST.MF`.

```
Manifest-Version: 1.0
Main-Class: B
```

We can see that the main class is specified. The command for creating JAR files is `jar`, the pattern for using it is as follows.

```
$ jar cfm jar-file manifest-file input-file(s)
```

For example, we can create a JAR file `HelloJar.jar` as follows, using the Manifest file that we defined earlier.

```
$ jar cfm HelloJar.jar MANIFEST.MF p/A.class p/B.class
```

Note that we only include the class files, and not the source code (`A.java` and `B.java`). A JAR file can contain source code, but that is not necessary. Having the JAR file, it can be executed as follows.

```
$ java -jar HelloJar.jar
```

When the Java virtual machine (JVM), that is, the command `java`, runs a program, the class files are loaded. The *classpath* specifies where the JVM should look for the class files. Specifying the classpath may be required when several different libraries are used that are located in different JAR files. The classpath can be given using the flag `-cp` as the following illustrates.

```
$ java -cp lib1.jar:lib2.jar:. MainClass
```

The dot (`.`) means the current working directory. The JVM will look for class files in the JAR files `lib1.jar` and `lib2.jar` and in the current working directory. The standard classpath is the current working directory, which is equivalent to the following.

```
$ java -cp . MainClass
```

Often it is required to include the library files when compiling the source code as well. For example, suppose we have a class `MainClass` that uses classes from the JAR files `lib1.jar` and `lib2.jar`. Then when compiling the `MainClass.java` file, we need to include the JAR files in the classpath as follows.

```
$ javac -cp lib1.jar:lib2.jar MainClass.java
```

3.3 Further reading

- A Java language cheat sheet from Princeton:
  `http://introcs.cs.princeton.edu/java/11cheatsheet/`
- Oracle's Java tutorial: `https://docs.oracle.com/javase/tutorial/java/index.html`
- The Java standard libraries: `http://docs.oracle.com/javase/8/docs/api/index.html`
- About running `java` and `javac` from the command line:
  ```
  man java
  man javac
  ```
- JAR tutorial: `https://docs.oracle.com/javase/tutorial/deployment/jar/index.html`

3.4 Check your proficiency

Now check your proficiency with creating, compiling and running Java programs by going back to the beginning of section 3 and do the tasks there.

4 Learn more

To get quick help, use google. Often you find good answers at stack overflow, *the* forum for day-to-day programming problems and copy&paste solutions (`http://stackoverflow.com`).

If you would like to become really proficient at Unix, *the* book about Unix as a programming environment is

Brian W. Kernighan and Rob Pike
The Unix Programming Environment
Prentice-Hall, 1983
ISBN 978-013-937681-8

The authors were members of the original Unix development team. The book is still valid and highly recommended (and not bulky!).

Two very good text editors integrating well with the Unix philosophy are Emacs (`https://www.gnu.org/software/emacs/`) and VIM (`http://www.vim.org/about.php`). Both are modal editors; different modes are used to navigate, manipulate and select text. If trained in typewriting, and after some time to get used to the different commands, these editors are very powerful programming tools; in particular as they integrate the Unix command line. Emacs goes further by heavily exploiting the Lisp programming language for user developed editor extensions. VIM is more programming language independent and closer to Unix (it actually is an i**m**proved version of Vi, the grandfather/master of Unix text editors).