CS 301 Spring 2016
Lab February 5

Tools

Plan

This document gives some directions for using, installing, and configuring software. Some of this (as marked inline) is already installed on provided platforms. Installation instructions for some pieces assume Linux or Mac. Windows will work for most things until code generation (end of the TINY compiler, or about spring break for our large compiler).

More Mercurial

Mercurial Convenience Setup In the `cs301-labs` workspace directory you setup, use Emacs (or any text editor) to open up the file `.hg/hgrc`. It should look something like this:

```
[paths]
default = ssh://hg@bitbucket.org/yourusername/cs301-labs
```

Immediately below the "`default = ...`" line, add this line:

```
starters = ssh://hg@bitbucket.org/wellesleycs301/cs301-labs
```

Now, whenever you need to pull new starter code, you can use the command:

```
hg pull starters
```

in place of:

```
hg pull ssh://hg@bitbucket.org/wellesleycs301/cs301-labs
```

You can still use `hg pull` and `hg push` as usual to pull and push between your Bitbucket repository and your local repository. (Remember, `hg pull` does not change the contents of your local working copy, just your local repository. To bring new changes into your working copy, use `hg merge` or `hg update` as suggested by the output of `hg pull`.) For more Mercurial reference, see: `https://cs.wellesley.edu/~cs301/common/hg/hg.html`

MercurialEclipse. This plugin provides repository management directly in Scala IDE. If you are using the wx appliance, it is already installed. Otherwise, install the MercurialEclipse plugin (once):

1. Under Help, open Eclipse Marketplace.
2. Search for "mercurial".
3. Select and install MercurialEclipse 2.1. Choose Continue, accept the license, install the unsigned content (sigh), etc.
4. Restart Eclipse/Scala IDE.

Enable MercurialEclipse management for a project:

1. Right-click the project in the Package Explorer pane (on the left) and choose Team > Share Project...
2. Choose Mercurial and click Next.
3. This should find the repository containing your whole CS 301 workspace. Select Use existing repository in the parent directory and click Finish.

4. Repeat for each project in the workspace that you wish to manage with the repository. Projects created or imported in this workspace later will automatically have version control enabled.

Now you can manage the Mercurial repository from within Eclipse by right-clicking a resource in the Package Explorer pane using the Team submenu. MercurialEclipse also automatically renames files in the repository when you do Refactorings through Eclipse (very nice!). It is also easy to commit, push, and pull through the GUI. I tend to prefer the command-line `hg` tool for committing, pushing, and pulling, but I use the GUI for file management most of the time. If you mix the command line and the GUI like this, you may need to refresh the project in Eclipse to see the up-to-date repository state after some command-line operations. Use what works for you.

Eclipse Scala IDE

Scala IDE is a repackaged version of Eclipse with the Scala IDE plugin preinstalled. You can also install the plugin into Eclipse Luna (4.4). Other versions may be less stable.

Preferences. Most Preferences can be found under the Scala IDE menu on Mac and the Window menu on Linux. Here are a few of potential interest:

- **Show line numbers.** Toggle open the General group, toggle open its Editors subgroup, select Text Editors (select, don't toggle), and check the option for line numbers.
- **Change font or size.** Toggle open the General group, toggle open its Appearance subgroup, and select Colors and Fonts. In the open pane on the right, toggle open the Basic group, select `Text Font`, and click the Edit... button on the right.
- **Enable Emacs or default key bindings.** Toggle open the General group, select Keys, and choose the Scheme you want.

Navigation and Screen Space. By default, the Package Explorer pane on the left shows in "flat" view. You may prefer "hierarchical" view. Click the down-pointing triangle near the top of that pane to get a menu. Choose Package Presentation > Hierarchical.

Arrows on the right of the top toolbar act like cross-tab browser back/forward buttons. The left arrow with a * goes to the last edited location.

Minimize panes you use infrequently with the horizontal minimize button in the pane header. They will jump back out temporarily if you run an operation that displays info there (*e.g.*, the reference searches described above). Click the individual pane icons to get the pane temporarily. Click the two-stacked-windows icon to restore the pane group.

Building, Cleaning, Refreshing. The Scala IDE automatically compiles each time you save, but it is not very fast. (You might beat it if you switch over to the command line to run something quickly.) A progress bar appears in the lower right corner while building. If you make changes to files in a project from outside the Scala IDE (including by running `hg`), right-click the project in the Package Explorer and choose Refresh (or select it and hit the F5 key). You might find that the automatic build process gets a little confused when you edit files externally. If things are still odd after refreshing, force a full recompilation of the whole project (or workspace) with Project > Clean....

Run Configurations. If you prefer to run things without leaving the Scala IDE, click the green Run button (or use the Run menu) and choose Scala Application. Choose the main class (the `class` or `object` containing the `main` method to run). Give the fully qualified name of this class (including its package). In the Arguments tab, give any arguments needed. Eclipse remembers some recent Run configurations within each project and runs the most recently used by default.

Searching. Normal text search is possible, but the IDE provides more structured semantic search that is quite useful. Select the name of a variable, field, method, class, etc. in the editor and hit the F3 key to jump to its declaration. Searching for references to a name is also possible. Eclipse supports this for Java; a plugin is required for Scala IDE. To install:

1. Open Help > Install New Software....
2. Select the Scala IDE site from the top drop-down menu (Work with: "type or select a site").
3. Toggle open Scala IDE plugins (incubation)
4. Check the box for Scala Search.
5. Click Next, Next, accept the Apache license, click Finish.
6. When prompted, restart Scala IDE.

Now you can search for occurrences of/references to a name by selecting it and right-clicking to choose Find Occurrences. (Or type Shift-Command-Option-R on a Mac or Control-Shift-Alt-R on Linux.) Double-click a listed occurrences in the resulting menu to jump there.

Refactoring. Scala IDE provides a few refactoring tools, but the most-used refactorings are for "simple" renaming tasks. Find-and-replace can work, but using the refactoring does *semantic* find-and-replace of names of local variables, fields, methods, classes, etc. Highlight the name to change. Then, right-click and choose Refactor > Rename. Alternatively, type Command-Option-R on a Mac or Shift-Alt-R on Linux.

Command-line Scala

No installation is required on provided platforms.

If you wish to use Scala on the command-line and do not already have it, install Scala 2.11.7. If you are using one of the provided platforms, this is already installed.

On Linux, the easiest way is to use your distribution's package manager, but check the version – we need Scala 2.11.7.

On a Mac, the easiest way is to use Homebrew (if you already have that installed): `brew install scala`.

Alternatively, on Linux or Mac, you can download Scala from `http://scala-lang.org/download/`. Extract the file anywhere. It should create a directory called `scala-2.11.7`. You can now run `scala` (or any of the associated command-line tools) with the command `the/path/to/your/scala-2.11.7/bin/scala`. To make this available as `scala`, edit `~/.bash_profile` to add the line:

```
export PATH="$PATH:/the/absolute/path/to/your/scala-2.11.7/bin"
```

Open a new shell to load this configuration.

The same should be possible for Windows, replacing the paths and path configuration appropriately.

GCC and GDB / Clang and LLDB

No installation is required on provided platforms. We do not need this immediately.

When we get to code generation, our compiler will emit assembly code that we then assemble and link into an executable. For this last step, we will use GCC on Linux or Clang/LLVM on Mac. For debugging generated code we will use GDB (Linux) or LLDB (Mac).

On Linux, use your package manager to install `gcc` and `gdb` if not already installed.

On Mac, run `xcode-select --install` to install.