Introduction to Course Infrastructure & Java

During this course you will become familiar with several popular (and industry-relevant) software engineering tools including Eclipse, Git, Gradle, Travis CI, FindBugs, and Checkstyle, in addition to Java. In this recitation you will make sure your environment is set up correctly. If you have problems with the course infrastructure, reach out to the course staff as soon as possible through Piazza or office hours.

Git and GitHub

Git is a distributed version control system commonly used for large software projects, and GitHub is a hosting service for Git repositories. We will be using GitHub and Git to distribute homework assignments, for you to turn in your homework, and for us to give you grades and other feedback on your work. The basic idea is that you will clone – or make a copy of – your GitHub repository on your local computer. You can then pull changes from GitHub to receive our feedback and any new homework assignments, work locally on your own computer to complete your homework, and commit and push your completed homework back to GitHub, so we can grade it.

Setting up your repository

To set up your repository, sign up for a GitHub account; you may use your existing GitHub account, if you have one. Then fill out the web form here:

`http://garrod.isri.cmu.edu/214/registration`

After filling out that form, confirm your email. Upon confirmation, a GitHub repository will be set up for you and you will receive an email from GitHub asking you to join the 15214 organization. If you do not get this email, go to https://github.com/CMU-15-214/ and join the organization.

Make an initial clone of your Git repository with:

```
$ git clone https://github.com/CMU-15-214/your-andrew-id.git
```

where *your-andrew-id* is your Andrew ID. This will create a directory on your local computer, with your Andrew ID as the name. This directory is the copy of your Git repository in which you will work. Note that graphical frontends for Git are available, such as GitHub Desktop, but we recommend to use the command line tools.

Retrieving assignments and grades

If you have already cloned your Git repository to your local computer, you can pull changes from GitHub (to receive new assignments or grades, or work you've committed from another clone of your repository) with:

```
$ git pull
```

Build and Test Automation

Gradle is a build tool that automates building, deploying, and testing projects and takes the best features from both Ant and Maven. Gradle allows you to build, analyze, and execute your code from the command line. It will later be useful to automatically manage dependencies as well. Travis CI is a web service that automatically builds your code and runs tests when you push to GitHub, sending you an email if the tests fail (if you have set up your email for git commits). These tools enable you to more-easily maintain the integrity of your code, ensuring that new code does not break your project.

Checkstyle is a development tool to point out when Java code does not adhere to common conventions. It automates the process of checking for common stylistic mistakes such as the use of magic numbers. For this course, we will be following a subset of the Sun Code Conventions which can be viewed here. Checkstyle is set up to run automatically with Gradle and will cause your build to fail when the guidelines are not followed. You can run Gradle locally or wait for a message from Travis-CI. There are also plugins for IDEs as Eclipse and IntelliJ IDEA if you prefer immediate feedback in the IDE. Checkstyle is not an autograding tool; it is only meant to help you to avoid certain common mistakes.

Development Environments

Integrated Development Environments (IDE) tend to provide a much better experience than classic text editors for writing Java code. There are several IDEs available, but Eclipse and IntelliJ IDEA tend to be the most common ones. In addition to the Gradle build files, we will distribute setups for Eclipse projects (in Eclipse use "Import – Existing Projects into Workspace"), but you can easily import those with other IDEs as well. It is up to you which IDE you use, if any.

Exercise: Java Practice

After you have imported the "rec01" project to your preferred IDE, examine the `Example` and `Main` classes. Complete the `printList` method for the `Example` class.

Turning in your work

After you are done working within a clone of your repository, you can turn in your work with

```
$ git add file1 file2 dir1 ...
$ git commit -m "Completed recitation 01"
$ git push
```

where *file1*, *file2*, *dir1* and so on are the names of files and directories you have added or changed and the commit message (after the `-m`) is an arbitrary message to describe your work.

1. The command (`git add`) instructs git to track changes to a set of files in your clone; this is called adding the files to your *staging area*. If you pass a directory name to `git add` then all the files added or changed in that directory and all subdirectories (recursively) will be tracked and staged.[1] You can check what files are staged with the `git status` command.
2. The command (`git commit`) records all the locally-tracked changes as a new version of the repository, along with a message that describes the new version. You can check all of the recent commits on your machine with the `git log` command.
3. The command (`git push`) records the most-recent committed version to the remote server, your repository on GitHub. GitHub will then automatically trigger a build on Travis-CI.

Your work is not turned in unless you have completed all three steps. Check on GitHub whether all your files are correctly pushed.

Each new version is essentially a *local* checkpoint of your work, which you can turn in when you next push your repository to GitHub. If you push your repository to GitHub but have not staged and committed your changes, those changes will not be pushed to GitHub.

Whenever you have finished a feature of your homework, though, it is a good practice to commit it to your repository by doing the above. You should commit often and use helpful commit messages. It is common to commit multiple versions locally before pushing your work, although you might want to periodically push your work to GitHub even if your homework is not complete because this essentially makes a backup copy of your work.

Also, if you attempt to push your repository to GitHub but the GitHub repository has changed since you last ran `git pull`, your push will fail. To fix this you just need to pull the other changes from GitHub (using `git pull`) and attempt your push again.

[1] Remember that `.` denotes the current directory. So, (`git add .`) will add all files in the current directory to the staging area.

When you are done pushing your work to GitHub, you should always check GitHub to confirm your expected files are there. Alternatively, you can create a new clone of your repository (using `git clone`) in a new location on your computer, and test your solution in that new location. This method allows you to test exactly what the TAs will test when they clone your repository from GitHub.

Don't push changes to a homework subfolder after the deadline (unless you intend to spend part of your "late" budget). We will use the GitHub timestamp of the latest `git push` event to determine if you were on time.

Additional resources

Here are some resources if you are interested in learning more about git. We recommend anyone who is going into any industrial software development company to at least try the 15-minute tutorial on git. Version control is a very powerful tool in software development, and mastering it will remove many headaches that you may encounter otherwise.

- 15-Minute Tutorial: https://try.github.io/
- Pro-Git Book (free): http://git-scm.com/book/en/v2/
- Github Game (free): http://pcottle.github.io/learnGitBranching/