

SWEN20003  
Object Oriented Software Development

Software Tools and Bagel

**Shanika Karunasekera**  
karus@unimelb.edu.au

University of Melbourne  
© University of Melbourne 2020

# The Road So Far

- OOP Foundations
  - ▶ Subject Introduction
  - ▶ A Quick Tour of Java
  - ▶ Classes and Objects
  - ▶ Arrays and Strings
  - ▶ Input and Output

# Lecture Objectives

After this lecture you will be able to:

- Use **git** for **software version control**
- Use **Maven** to **manage software builds**
- Use the **IntelliJ Debugger** to find program bugs
- Use the software package **Bagel** (Basic Academic Graphical Engine Library)
- Contribute to **Open Source Software** (OSS) projects

# Version Control with git

# Introduction

Software version control:

- A systematic way to manage concurrent versions of software artefacts - documents, source code, data etc.

Why do we need software version control?

- As an individual, version control allows you to keep a version of the software saved, so that you can revert back to the previous version if necessary.
- In a team software development setting, version control allows developers to write and test code locally before including that to code base.

We will introduce you to the basics of software version control with `git`.

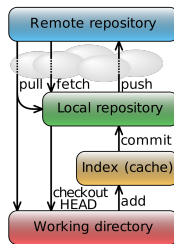
# What is git?

- git is a type of *version control* system.
- It is probably the most commonly-used modern system (approximately 70%).
- Created by Linus Torvalds, who also created Linux.



# The git model - repositories, working directory and staging

- **Repository:** A collection of files to be stored by the version control system, together with some metadata to keep track of them.
- **Remote repository:** A repository maintained in a remote server (e.g. GitHub, Bitbucket, GitLab).
- **Local repository:** A repository maintained in the local directory - in `.git` sub directory.
- **Index (Staging Area):** A conceptual place for staging the changes - in `.git` sub directory.
- **Working directory:** The local directory where the files are kept and modified.



**Figure:** Git Model

([https://commons.wikimedia.org/wiki/File:Git\\_data\\_flow\\_simplified.svg](https://commons.wikimedia.org/wiki/File:Git_data_flow_simplified.svg))

# The git model - branching

- **Master:** the current “known good” codebase
- **Branch:** a version of the codebase being worked on (e.g. security-support, UI-improvements)
- **Commit:** a snapshot of a branch

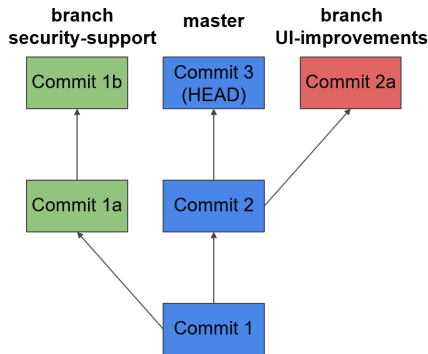


Figure: Git Model - Branching



# The git model - merging

- **Merging:** incorporating the changes in one branch to another
- **Merge request:** a request to merge completed feature branches into the master branch (also referred to as a pull request)

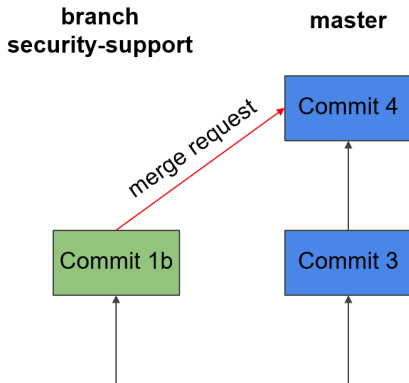


Figure: Git Model - Merging

# Using git

You will be required to use git for managing your project code in this subject.

You will use the remote GitLab repository (installed in a MSE server), available at `gitlab.eng.unimelb.edu.au`. - each of you will have an account there.

There are two different ways to interact with git:

- Graphical User Interface - GUI
  - ▶ There are GUIs specifically designed as git clients and also IDEs such as IntelliJ have git integrated to its GUI
- Command-line Interface - CLI

# Using git

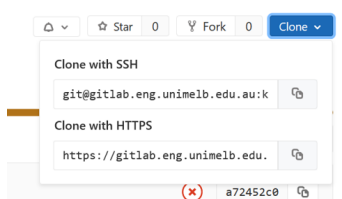
We will next look at how to create a new repository in GitLab for an existing project (we have already created projects for you).

- ➊ **Step 0:** Get the remote repository URL
- ➋ **Step 1:** Make a local copy of the repository
- ➌ **Step 2:** Stage the initial files
- ➍ **Step 3:** Update the local repository
- ➎ **Step 4:** Update the remote repository

# Using git

## Step 0: Get the remote repository URL

- 1 Login to `gitlab.eng.unimelb.edu.au` using your university email address
- 2 You will see the projects we have already created for you. e.g. `[user-name]-workshops` - they are uninitialized
- 3 Copy the `[clone URL]` from GitLab by expanding the Clone button on the page (https option would require a key)



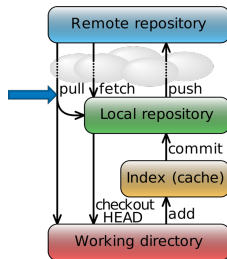
# Using git

## Step 1: Make a local copy of the repository

- 1 Open a command window on your local machine (git must be installed in the local machine)
- 2 Change directory to where you want to create the local copy
- 3 Make a local copy using the following command:

```
git clone [clone URL]
```

clone (instead of pull) is used to create a local copy of a remote repository.



## Example

```
git clone git@gitlab.eng.unimelb.edu.au:swen20003-s2-2020/karus/demo.git
```

# Using git

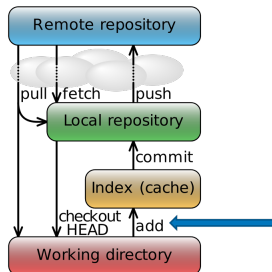
## Step 2: Stage the initial files

- 1 Copy the files to the directory (demo in the above example).
- 2 Change to the directory (demo in the above example).
- 3 Stage all the files in the directory using the following command:

```
git add .
```

The following command will stage specific files:

```
git add [file name]
```



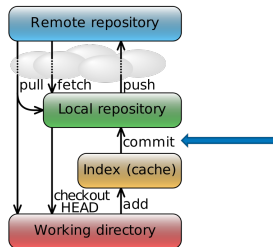
# Using git

## Step 3: Update the local repository

- Add the files to local repository using the following command:

```
git commit -m "message"
```

Note: Ensure that the “message” is a meaningful to describe the changes in the particular commit.

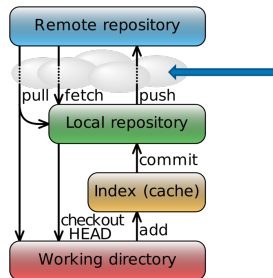


# Using git

## Step 4: Update the remote repository

- Add the changes to the master branch of the remote repository using the following command.

```
git push -u origin master
```





# Using git

Next time you want to add, delete or modify files/folders in the directories following are the steps to follow.

❶ Add/remove/change files in the folder as needed.

❷ Stage the files by using the following command:

```
git add [filename]
```

This will stage the files one at a time - or you can stage a group of files as needed.

❸ Commit the staged files to the repository using:

```
git commit -m "Commit message"
```

❹ Add the files to the remote repository using:

```
git push
```

# Using git - Useful Commands

Here are some commands that are good to know in your git adventures.

- `git status`: shows you the current status of your commit, including which files have been modified, and which of those modifications are staged.
- `git log`: shows you the history of the repository, including any previous commits
- `git branch`: shows the current branch you are in
- `git reset --hard`: If you accidentally delete some files, or break everything by changing the code, you can easily go back to the previous committed version using

# But wait, there's more...

This has been a very quick introduction to git! There are many more advanced features that make *collaborating* on projects much easier, such as branching and merging. I have only focused on what you need to get started with your own repositories.

# **Build Management with Maven**

# Build Management






Build management/automation tools aim to “automate” the process of building (compiling) software - e.g. make, Apache Ant.

A quick Google search brings up the following:

## List Of The Top Build Automation Tools

Enlisted below are the most popular Build Software products that are used worldwide.

### Comparison of the Best Automated Build Deployment Software

Automation Tools	Best For	One Line Description	Free Trial	Price
<b>Jenkins</b>  <b>Jenkins</b>	Small to Large Businesses	Automation server used to Build, Deploy, and Automate any project.	No	Free
<b>Maven</b> 	Small to Large Businesses	Project management and comprehension tool.	No	Free
<b>Gradle</b> 	Small to Large Businesses	Build Tool	30 days	Get a quote
<b>Travis CI</b> 	Small to Large Businesses	Sync GitHub projects and test.	For 100 builds	Free for open source projects. Bootstrap: \$69/month Startup: \$129/month Small Business: \$249/month Premium: \$489/month
<b>Bamboo</b> 	Small to Large Businesses	Continuous Integration & Deployment Build Server	30 days	Small Teams: \$10 for 10 jobs. Growing Teams: \$1100 for unlimited jobs.

# What is Maven?

Maven is a popular software build management tool.

Maven simplifies the process to build software by automatically importing the required libraries and dependencies from external repositories.



In Maven, the project structure and contents are declared in an xml file, Project Object Model (POM) - `pom.xml`.

# Using Maven

You will be required to use Maven to build your projects - to access the required packages to build your project.

However, for this subject, you do not require an in-depth understanding of the details of Maven.

We will provide you the required directory structure and the files including the `pom.xml`, and the instructions to build it.

You can import this to IntelliJ as a Maven project and you will be ready to start developing!

We will practice this in Workshop 4 (week 5).

# Using the IntelliJ Debugger



# Debugging

Debugging refers to finding and fixing bugs in software.

What techniques do you currently use for debugging?

- Randomly change code and hope it works?
- “Debug by printf”?
- Or the student favourite: ask the teacher!

Fortunately, there's a better way...

# Advanced Debugging using Debuggers

Debuggers let you walk through a program step-by-step.

They allow runtime examination of programs.

We will look at some of the features of the IntelliJ Debugger.

# IntelliJ Debugger Capabilities

## Keyword

*Breakpoint:* A predetermined line where your code will *pause*, allowing you to inspect its state; the contents of variables, any active methods, etc.

## Keyword

*Watch Expression:* An expression that you want to see the value of.

## Keyword

*Step over:* Jump to the next line of code, stepping over any method calls in this line.

## Keyword

*Step into/ Step out of:* Jump into the code of the method in the particular line and jump out of the method call.

A good demonstration of the IntelliJ Debugger, which shows the most useful capabilities is available [here](#).

# Bagel

# Introduction

**Bagel: Basic Academic Graphical Engine Library** is a custom graphics package for Java.

- Developed for SWEN2003 by Eleanor McMurtry, the Head Tutor for the subject
- Uses an existing library LWJGL
- Previously we used an open source graphics package, Slick, which was too complex and was getting outdated.

You will be required to use Bagel for your projects:

- Learning how to use software packages is an important skill you will have to develop as a software engineer.
- We will guide you through the process.

# Overview of Bagel

In a Bagel application, the main method looks as follows:

```
1  import bagel.*;
2
3  public class BagelTest extends AbstractGame {
4      // The entry point for the program.
5      public static void main(String[] args) {
6          BagelTest game = new BagelTest();
7          game.run();
8      }
9  }
```

- **Line 3:** Defines the class - you will learn about `extends` next week.
- **Line 6:** Bagel main class instantiates and object of the same type. calls
- **Line 7:** Calls the `run()` method in the `BagelTest` class.

Where is the `run()` method?

# Overview of Bagel

The update method:

```
1  //Performs a state update.
2  @Override
3  public void update(Input input) {
4      // Your code goes here
5  }
```

About 60 times per second, Bagel:

- Clears the window of images, leaving a uniform colour
- Checks for keyboard or mouse input
- Calls the update method
- Input class - will explain in the coming slides

You as a developer, have to write code in the update method to perform the required logic to update the state.

# Overview of Bagel

The Image class:

```
1    Image bagel = new Image("res/bagel.png");  
2    bagel.draw(Window.getWidth() / 2.0, Window.getHeight() / 2.0);
```

- Creates an Image object (Line 1) and calls the draw() method (Line 2)
- An image that can be drawn on the screen
- The draw() method should be called from the update() method
- Window class contains static methods to work with the display window
- You will need to refer to the Bagel documentation for more details about the classes and methods



# Overview of Bagel

## The Input class:

- Passed as an argument to the `update()` method
- Can be used to work with the keyboard and mouse
- Has methods to check key press and mouse click events

```
1         if (input.isDown(Keys.DOWN)) {  
2             y += speed;  
3         }  
4  
5         if (input.wasPressed(Keys.ESCAPE)) {  
6             Window.close();  
7         }
```

# Overview of Bagel

Putting it all together:

```
1      import bagel.*;
2      // Simple Bagel demonstration
3      public class BagelTest extends AbstractGame {
4          private Image smiley;
5          private float x = 100,y=100;
6          public BagelTest() {
7              super(800, 600, "Bagel Demo");
8              smiley = new Image("res/smiley.png");
9          }
10         public static void main(String[] args) {
11             BagelTest game = new BagelTest();
12             game.run();
13         }
14         @Override
15         public void update(Input input) {
16             float speed = 0.5f;
17             if (input.isDown(Keys.LEFT)) { x -= speed; }
18             if (input.isDown(Keys.RIGHT)) { x += speed; }
19             if (input.isDown(Keys.UP)) { y -= speed; }
20             if (input.isDown(Keys.DOWN)) { y += speed; }
21             if (input.wasPressed(Keys.ESCAPE)) { Window.close(); }
22             smiley.draw(x, y);
23         }
24     }
```

# Overview of Bagel

For the full documentation of classes and methods in Bagel, see: [Bagel Documentation](#).

Also available on [Canvas](#).

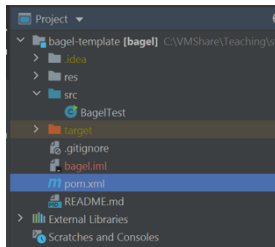
We will get you started on using Bagel in the next workshop.

# Using Bagel

- 1 Download the skeleton (e.g. `bagel-test.zip`) from Canvas and extract it - this will be a Maven project
- 2 Make a copy of it in the folder you are going to work on it.
  - ▶ For Workshop 4 and the projects you will be required to add this to git, so you will have to place it in the appropriate git repository folder
- 3 Import the project to IntelliJ using:

File -> New -> Project from Existing Sources

Select the appropriate folder and import as a Maven Project and you should be able to run the program.



# Open Sources Projects

# What is an Open-Source Project

## Keyword

*Open-Source:* Software for which the original source code is made freely available and may be redistributed and modified.

## How do they work?

All Open-Source projects are... well, open.

Anyone can contribute... So long as you follow the “rules” of the project.

Each project also comes with their own set of *tools*; some are mandatory (like a build system, or a project management tool), some are optional (like joining a Slack team).

# How do you start?

There are plenty of beginner-friendly entry points, such as:

- [First Timers Only](#)
- Jump in [here](#), or [here](#), or [here](#)

Try your hand at a simple, easily completed task. What's the worst that can happen?



# Why do you care?

Contributing to open-source projects is **one of the best ways to demonstrate employability**.

Why?

Because making a contribution, big or small, requires a lot of things:

- High quality code
- Understanding a complex application
- Learning new, unfamiliar skills
- Working in a distributed team
- Following the *rules*, *expectations*, and *guidelines* of the project

**Google**

Google is literally a developer's best friend.

In no universe will you get hired for a job where you understand or are familiar with 100% of the tools. You need to be able to *find your own solutions*.

How do you demonstrate you can do that?

- Contribute to an open-source project!
- Build your own projects, outside of the scope of university
- Do things you've never done before, that force you to learn new skills; then talk about it!

# Final Notes

What should you take from today's lecture?

- git is useful, and you should learn it and will have to use it for our projects
- There are a million and one tools you can use to make projects easier, particularly in teams; get familiar with a few of them
- The debugger is your friend
- How to user Bagel
- Open-source projects are excellent gateways for skill-building; give them a go
- Learn. How. To. Google.

# Lecture Objectives

After this lecture you will be able to:

- Use **git** for **software version control**
- Use **Maven** to **manage software builds**
- Use the **IntelliJ Debugger** to find program bugs
- Use the software package **Bagel** (Basic Academic Graphical Engine Library)
- Contribute to **Open Source Software** (OSS) projects