JC2002 Java Programming

Day 1: Introduction (CS)

Monday, 30 October

JC2002 Java Programming

Day 1, Session 1: Course introduction and practicalities

Icebreaker activities

- Who am I?
- How familiar are you with Java and programming in general?
- What do you expect from this course?

Course arrangements

- **Lectures** (theory sessions) on Mon, Tue, Wed, Fri
- **Practical sessions** on Mon, Tue, Thu
- **Practical assignment**, 30% of the assessment
  - Assignment will be released in the 3rd week
  - Submission in Codio (deadline: December 10)
- **Exam**, 70% of the assessment
  - Paper-based closed-book exam

Course topics

- **Week 1:** Java fundamentals
  - Using the basic Java tools, version control, testing
  - Java syntax, conditional structures, loops, classes, objects, methods
- **Week 2:** Object oriented programming, graphical user interfaces
  - Memory concepts, exceptions
  - Using Swing for implementing graphical user interfaces in Java
- **Week 3:** Concurrency and data structures
  - Multithreading and memory concepts
  - String operations, collections
- **Week 4:** Advanced topics in Java
  - File handling, database programming, etc.

Course topics – not only Java

- Professional software development practices
    - Version source control to keep track of development
    - Use of testing, and automated testing to ease development
    - Linux command line, read documentation, etc.

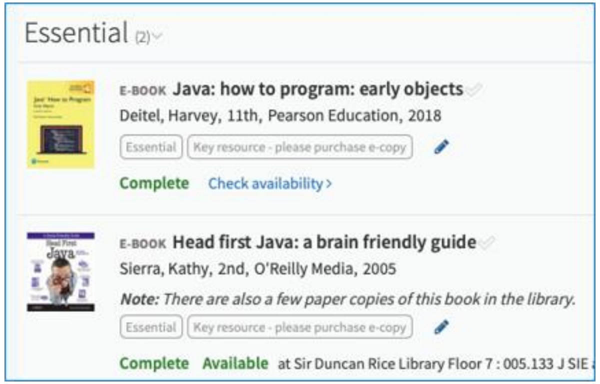
Java Platform, Standard Edition Tools Reference

javadoc

```
howtogeek@ubuntu:~$
```

Read the course textbooks

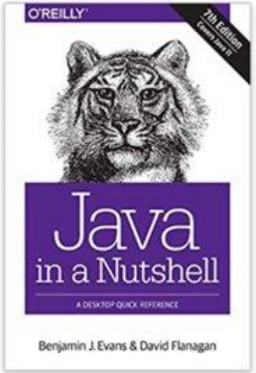
Lectures are only part of the process. You also need to read about the language!



Essential (2)

E-BOOK **Java: how to program: early objects**
Deitel, Harvey, 11th, Pearson Education, 2018
Essential | Key resource - please purchase e-copy
Complete Check availability >

E-BOOK **Head first Java: a brain friendly guide**
Sierra, Kathy, 2nd, O'Reilly Media, 2005
Note: There are also a few paper copies of this book in the library.
Essential | Key resource - please purchase e-copy
Complete Available at Sir Duncan Rice Library Floor 7 : 005.133 J SIE



- Course materials in library and online (We'll be adding more)
- Lots of materials in the library on software development

  -> both as books and online books

Assignment

- Programming assignment – 30% of the mark
- Exam – 70% of the mark
- Late submission policy
  - up to 24 hours late will incur a 10% penalty to your mark
  - up to 1 week late will incur a 25% penalty
  - Any hand-ins after that will be classified as no submission.
- Feedback
  - Feedback will be given no later than three weeks after you submit your work

Academic integrity

- **Plagiarism** is the unattributed use of other people’s work or ideas, in work submitted for assessment.
- **Collusion** occurs where there is unauthorised collaboration between students in the writing of an individual assignment.
- **“Contract cheating”** is a form of academic misconduct where a student submits work that has been produced by someone other than the student, whether this has been paid for or not.

See: https://www.abdn.ac.uk/students/academic-life/academic-integrity.php

Academic integrity

- Plagiarism

Reference and cite appropriately sources used in assessments and other learning activities.

Include direct quotes, if they are deemed appropriate by your subject area, acknowledged with quotation marks or indented, and remember to reference or cite your sources.

Paraphrase ideas from your sources using your own words, but always reference or cite the original author(s).

Resist making only minor changes to the original text and using them in assignments.

Do not copy and paste into your work sections of material from the Internet, books, papers, reports, or other published work.

Do not self-plagiarise by re-using previous marked assessments and re-submitting them.

Reference or cite the sources of tables, figures, diagrams, and images used in assignments.

- Collusion

Do not collude with others when writing your own individual assignment.

Create a study plan and timetable for assessments, to avoid the pressure to collude.

Do not allow your peers to access, copy, or submit your assignment as their own work.

- "Contract cheating"

Never commission assignment outlines or plans from online sources.

Do not request anyone, including friends, family, or tutors to prepare an assignment for you, whether paid or unpaid.

Never engage with 'essay mills' or custom sites to prepare assessments of any kind.

1495 UNIVERSITY OF ABERDEEN

Plagiarism in reports

- If taking text verbatim, put it in quotes and *cite the source*
- Alternatively, paraphrase or summarize and *cite the source*
- As a rule of thumb, don’t do this for more than 3-4 sentences
- See https://integrity.mit.edu/handbook/writing-original-work for an excellent guide

Plagiarism in source code

- An excellent guide: https://integrity.mit.edu/handbook/writing-code
  - Never copy another student’s code
  - Don’t simply reuse code
- Cite the sources of code that you use within your code
- Ensure that any licensing conditions are adhered to

Automatic plagiarism detection

- Similarity checking tools spot modifications, such as:
  - Change variable, constant or function/procedure names
  - Simply replace language constructs
  - Change indentation or separation within source code
  - See http://www.upenn.edu/academicintegrity/ai_computercode.html for further examples

! In this course, plagiarism detection tool in Codio will be used to check the coursework assignments!

Ask for help if you are stuck

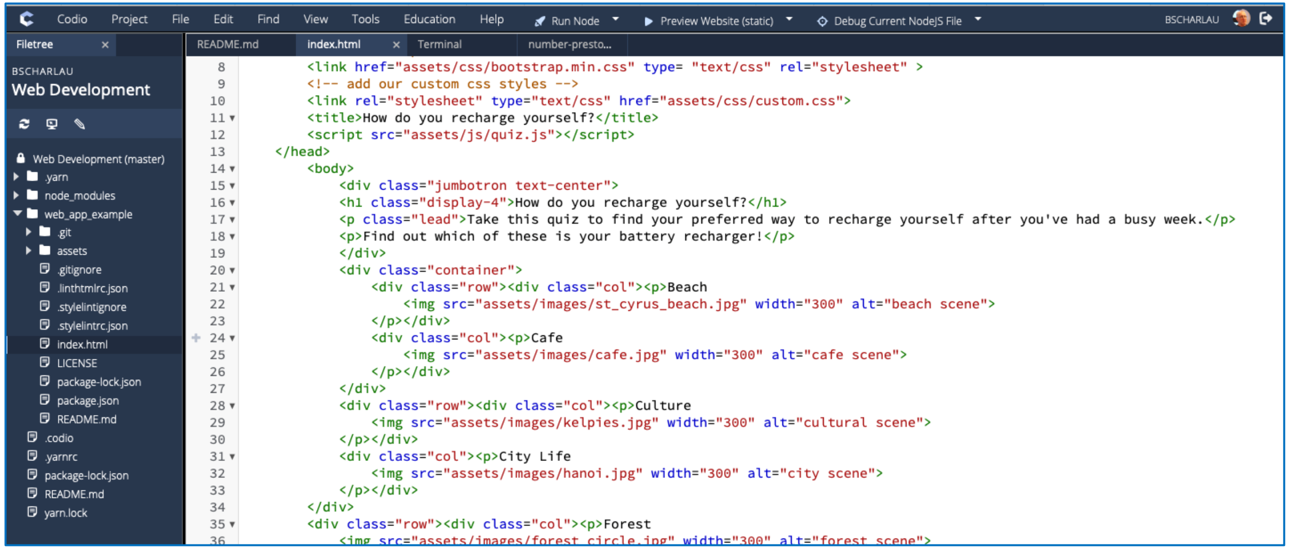
- If something is unclear, then ask someone or look it up
- If you still find it hard, get in touch with teacher
- Copying something from someone else, or via the internet, will not help you learn how to do the work

It is ALWAYS better to submit a poor assignment of your work with comments about what you tried, and struggle with, than to submit someone else’s work.

1495 UNIVERSITY OF ABERDEEN

Coding environment online: Codio

- Codio is integrated into this course via MyAberdeen



```
        <link href="assets/css/bootstrap.min.css" type= "text/css" rel="stylesheet" >
        <!-- add our custom css styles -->
        <link rel="stylesheet" type="text/css" href="assets/css/custom.css">
        <title>How do you recharge yourself?</title>
        <script src="assets/js/quiz.js"></script>
    </head>
        <body>
            <div class="jumbotron text-center">
            <h1 class="display-4">How do you recharge yourself?</h1>
            <p class="lead">Take this quiz to find your preferred way to recharge yourself after you've had a busy week.</p>
            <p>Find out which of these is your battery recharger!</p>
            </div>
            <div class="container">
                <div class="row"><div class="col"><p>Beach
                    <img src="assets/images/st_cyrus_beach.jpg" width="300" alt="beach scene">
                </p></div>
                <div class="col"><p>Cafe
                    <img src="assets/images/cafe.jpg" width="300" alt="cafe scene">
                </p></div>
            </div>
            <div class="row"><div class="col"><p>Culture
                <img src="assets/images/kelpies.jpg" width="300" alt="cultural scene">
            </p></div>
            <div class="col"><p>City Life
                <img src="assets/images/hanoi.jpg" width="300" alt="city scene">
            </p></div>
        </div>
        <div class="row"><div class="col"><p>Forest
            <img src="assets/images/forest_circle.jpg" width="300" alt="forest scene">
```

Version control

- Keeping code on local machines only is unsafe (disk failure)
- It is hard to work as a team if the members are working on their own versions of the code
- The solution: ***version control***
  - Version control allows to make remote copies of your code for sharing the code and rolling back to earlier versions easily
  - In this course, we will use **Git** for version control and as way to start working with remote repositories, such as **GitHub**

Questions, comments?

1495 UNIVERSITY OF ABERDEEN

JC2002 Java Programming

Day 1, Session 2: Introduction to Java language

Why Java?

- Java is one of the world’s most widely used computer programming languages
- For many organizations, the preferred language for meeting their enterprise programming needs is Java
- According to Oracle’s 2016 JavaOne conference keynote presentation (http://bit.ly/JavaOne2016Keynote), there are now 10 million Java developers worldwide and Java runs on 15 billion devices, including two billion vehicles and 350 million medical devices
- Android is a flavor of Java

Java history

- Java is an old language
  - Sun Microsystems in 1991 funded an internal corporate research project led by James Gosling, which resulted in a C++-based object-oriented programming language that Sun called Java
- The Internet helped Java grow
  - Java drew the attention of the business community because of the phenomenal interest in the Internet
  - Java programs run on a great variety of computer systems and computer-controlled devices (“*write once, run anywhere*”: details will be introduced later)
  - Now used to develop large-scale enterprise applications, to enhance the functionality of web servers, to provide applications for consumer devices, to develop robotics software and for many other purposes

Java versions

- There are different JAVA versions in use; the latest release of Standard Edition (SE) Java development kit (JDK) is 20
    - The latest version with long term support (LTS) is JDK 17
    - Not all versions are equal; this course mostly based on Java 11 with long term support (LTS)
    - For basics concepts in this course, any code for version from Java 8 will look almost identical
- Older Java RE runtimes may not run code written for newer versions and newer runtimes may be missing libraries required by programs written for older programs!

Java criticism

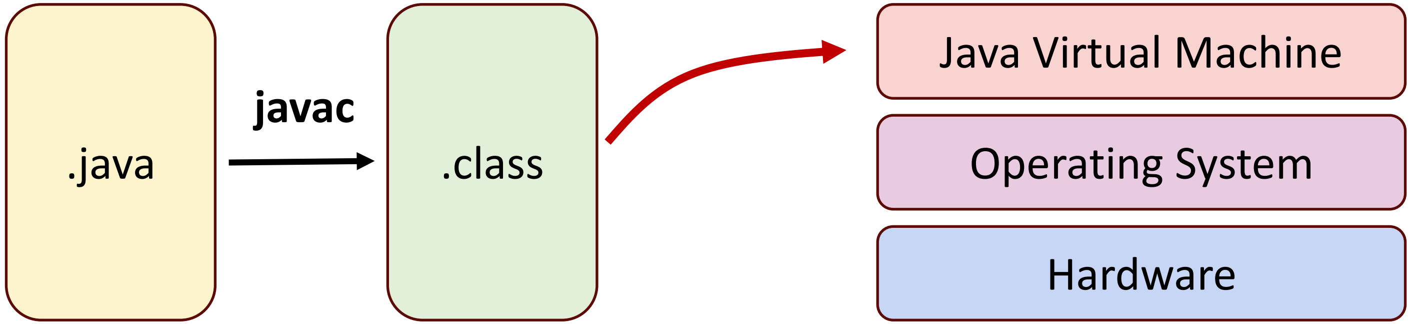
- Too verbose
  - However, easier to read!
  - The extra verbosity can be a benefit when you are responding to an outage call, or when you need to maintain and patch code that was written by developers who have long since moved on
- Slow to change
  - The new language features that have arrived in recent versions are a significant step toward addressing the most common complaints about missing features
- Low performance
  - True for the very early releases, but not longer a constraint

Java criticism (continues)

- Too verbose
- Slow to change
- Low performance
- Security concerns
  - During 2013, there were a lot of security vulnerabilities in the Java platform, which caused the release date of Java 8 to be pushed back
  - Many of these vulnerabilities involved the desktop and GUI components of the Java system, and do not affect websites or other server side code written in Java
- Too corporate
  - Actually, Java is a widely used language for open-source software projects

Java Virtual Machine (JVM)

- The JVM is a program that provides the “**runtime environment” (or executed environment)** necessary for Java programs to execute



- To compile .java file to .class file, use `javac Program.java`
- To run .class file, use `java <arguments> Program`

Benefits of JVM

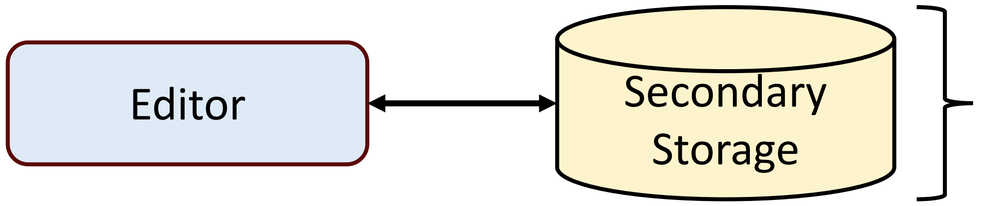
- Comprise a container for application code to run inside
- Provide a secure and reliable execution environment (as compared to C/C++)
- Take memory management out of the hands of developers
- Provide a cross-platform execution environment, i.e., “write once, run anywhere” (WORA)
- Make use of runtime information to self-manage, i.e., just-in-time (JIT) compilation

Typical Java development environment

- Normally, there are five phases in Java program development:
  - Phase 1: **Edit** the code
  - Phase 2: **Compile** to *bytecode*
  - Phase 3: **Load** the bytecode
  - Phase 4: **Verify** the bytecode
  - Phase 5: **Execute** the bytecode

Phase 1: creating a program

- Phase 1 consists of editing a file with an *editor program* (editor)
- Using the editor, you:
  - Type a Java program (source code)
  - Make any necessary corrections
  - Save it on a secondary storage device



Program is created in an editor and stored in a file with name ending `.java`

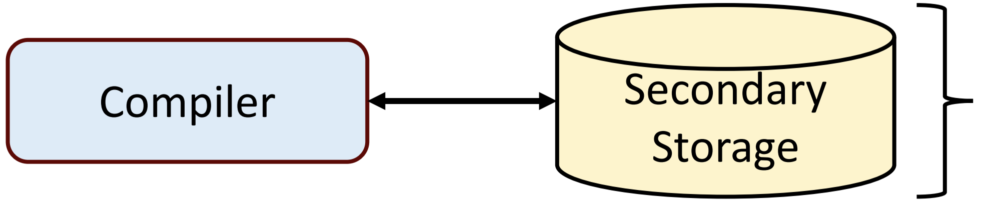
Phase 1: editing the program file

- You can edit source code file with any text editor (Vim, Notepad, TextEdit etc.)
- You can also use an *Integrated Development Environment* (IDE)
  - Provides tools to support software development process, such as editor, debuggers for locating logic errors, etc.
  - The most popular Java IDEs include:
    - Eclipse (http://www.eclipse.org)
    - IntelliJ IDEA (http://www.jetbrains.com)
    - NetBeans (http://www.netbeans.org)

Phase 2: compile a Java program

- Use the command `javac` (the Java compiler) to compile source code to a program
- To compile source file Welcome.java, you would type:

```
javac Welcome.java
```



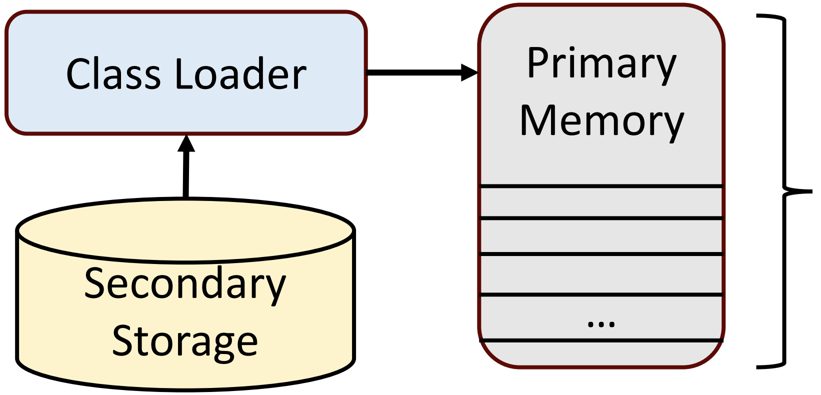
Compiler creates bytecodes and stores them in a file ending in `.class`

Phase 2: compiled bytecodes

- Java compiler translates Java source code into bytecodes that represent the tasks to execute in the execution phase
  - VMs can hide the underlying operating system and hardware from the programs that interact with it: if the same VM is implemented on many computer platforms, applications written for that type of VM can be used on all those platforms
  - The JVM —a part of the JDK and the foundation of the Java platform—executes bytecode
- So, Java's bytecodes are **portable**, the same bytecode instructions can execute on any platform containing a JVM that understands the version of Java in which the bytecodes were compiled

Phase 3: load program into memory

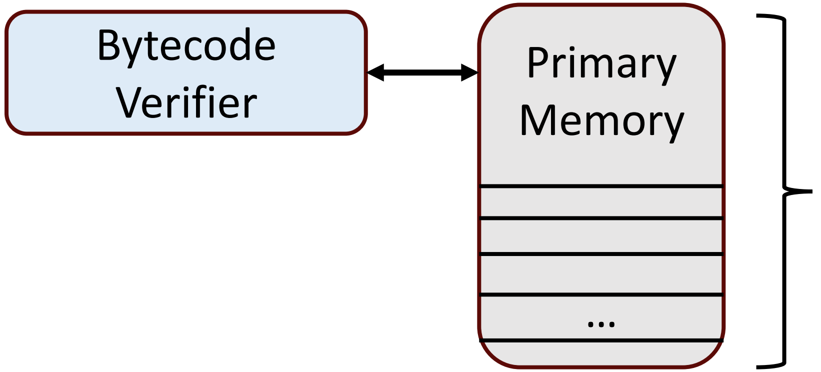
- The JVM places the program in memory to execute it — this is known as loading
- The class loader takes the `.class` files containing the program's bytecodes and transfers them to primary memory. It also loads any of the `.class` files provided by Java that your program uses



Class loader reads bytecodes from `.class` files and puts bytecodes in memory

Phase 4: verify the bytecode

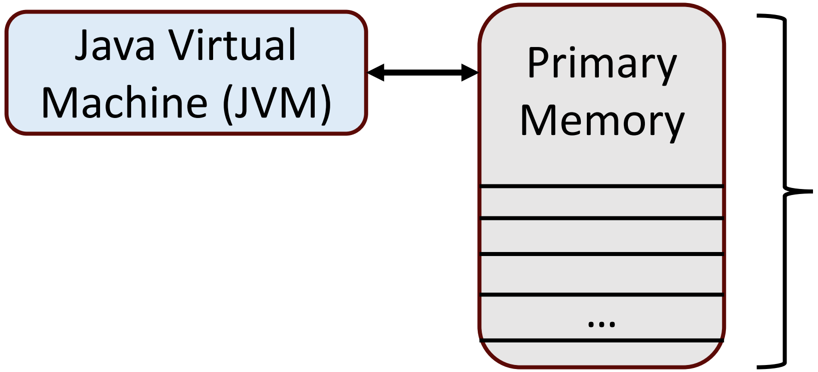
- As the classes are loaded, the bytecode verifier examines their bytecodes to ensure that they are valid and do not violate Java's security restrictions
    - Java enforces strong security to make sure that Java programs do not damage your files or your system (as e.g., computer viruses)



Bytecode verifier confirms that all bytecodes are valid and do not violate security restrictions

Phase 5: execution

- The JVM executes the program’s bytecodes
  - Today’s JVMs typically execute bytecodes using a combination of interpretation and so-called *just-in-time* (JIT) compilation
  - With JIT, the JVM can analyse the bytecodes as they’re interpreted, searching for hot spots – bytecodes that execute frequently



To execute the program, the JVM reads bytecodes and JIT compiles them into a language the computer understands

Phase 5: execution with just-in-time (JIT)

- A JIT compiler—such as Oracle's Java HotSpot™ compiler—translates the bytecodes into the computer's machine language
  - When the JVM encounters these compiled parts again, the faster machine-language code executes
- With JIT, Java programs go through two compilation phases
  - The first, in which source code is translated into bytecodes (for portability across JVMs on different computer platforms)
  - The second, in which, during execution, the bytecodes are translated into machine language for the actual computer on which the program executes

Common errors

- When using `javac`, an error message such as *"Bad command or filename"* or *"javac: command not found"* means that your Java software installation was not properly completed
  - Often, PATH environment variable was not set properly; carefully review the installation instructions if this happens
  - On some systems you need to reboot your computer after correcting PATH to make the change to take effect
- When using `java` to run a .class file, an error message such as *"java.lang.NoClassDefFoundError"* often means that Java CLASSPATH environment variable is not properly set

Java Classpath

- Java interpreter needs to know where to look for classes (`.jar` or `.class` files) that are not part of core Java
    - Classpath defines where to look for external bytecode files
- Two options to set Classpath:
    - Define CLASSPATH environment variable:

```
export CLASSPATH=.:/path/to/external/library.jar
```
(LINUX)

```
set CLASSPATH=.;/path/to/external/library.jar
```
(Windows)

- Use command line swith `–cp` or `–classpath` with `java` command:

```
java -classpath .:/path/to/external/library.jar ProgramName arg
java -cp .:/path/to/external/library.jar ProgramName arg
```

1495 UNIVERSITY OF ABERDEEN

Questions, comments?

JC2002 Java Programming

Day 1, Session 3: Version control

References for this session

- Evans, B. and Flanagan, D., 2018. *Java in a Nutshell: A Desktop Quick Reference 7th edition.* O'Reilly Media.
- Deitel, H., 2018. *Java How to Program, Early Objects, Global Edition, 11th Edition.* Pearson.
- Ben Lynn, Git Magic, 2007, Available online: http://www-cs-students.stanford.edu/~blynn/gitmagic/book.html
- Tom Preston-Werner, The Git Parable, 2009, Available online: https://tom.preston-werner.com/2009/05/19/the-git-parable.html
- Johan Herald, NDC TechTown, 2008, Available online: https://docs.google.com/presentation/d/1u0cM0r07iL9v7Myo6RWGWRR3o2IYlebDIayG8rMagVw/edit#slide=id.g3bcc4c1a94_0_6

JAR files

- Compiled JAVA programs (i.e., the `.class` files) can be packed into `.jar` files
- Third party libraries are often distributed as `.jar` files (via Maven)
    - We will look at Maven later when we start automating management of project dependencies
- You can also pack your own application as executable `.jar`
    - Running .jar files: `java -jar example.jar`
- More info on using `.jar` files: https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html

Version control

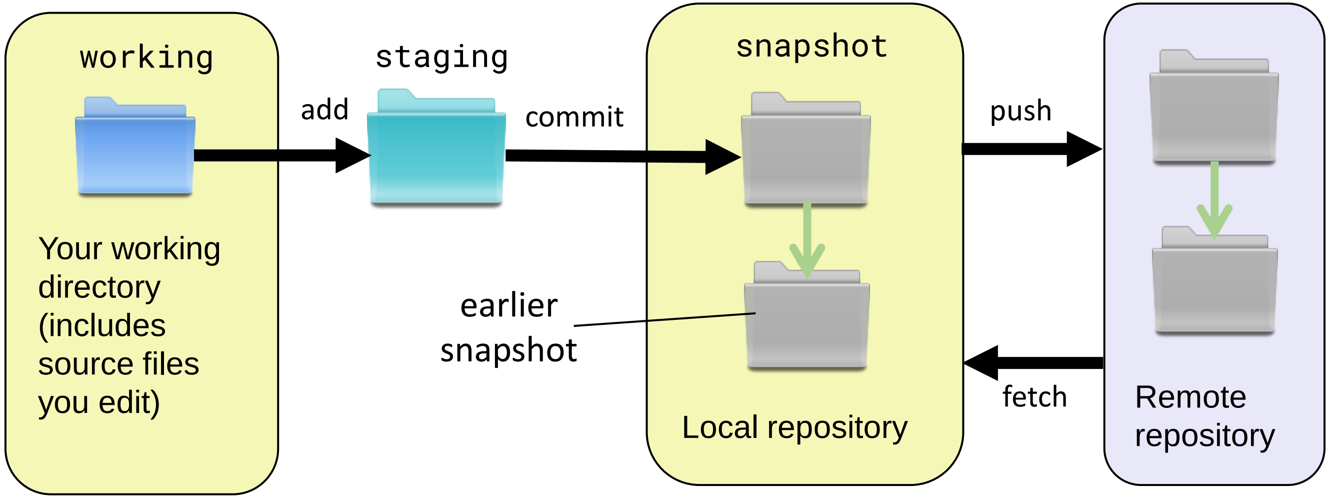
- Version control provides an undo for us so we can make changes with a safety net
- There are many types of version control:
  - Local (e.g., RCS)
  - Server based (e.g., Subversion)
  - Distributed (e.g., Git)
- On this course, we focus on Git

What is Git?

- An open-source distributed version control system
  - Original author Linus Torvalds
  - Created for the development of Linux Kernel
  - Most popular version control system today
  - Supported by many popular service providers such as *github.com*, *bitbucket.com*, etc.
- Git provides means for both:
  - A global “what if”
  - A global “undo” in our projects

Git snapshots

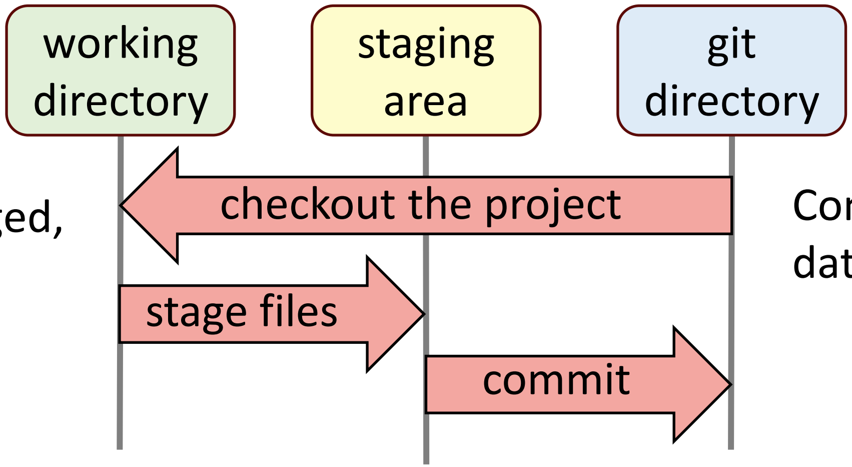
- In Git, snapshot is a record of the state of your project files at a specific point in time



Git areas of operation

- Git has three areas of operation: working directory, staging area, and git directory (repository)

Modified is changed, uncommitted file

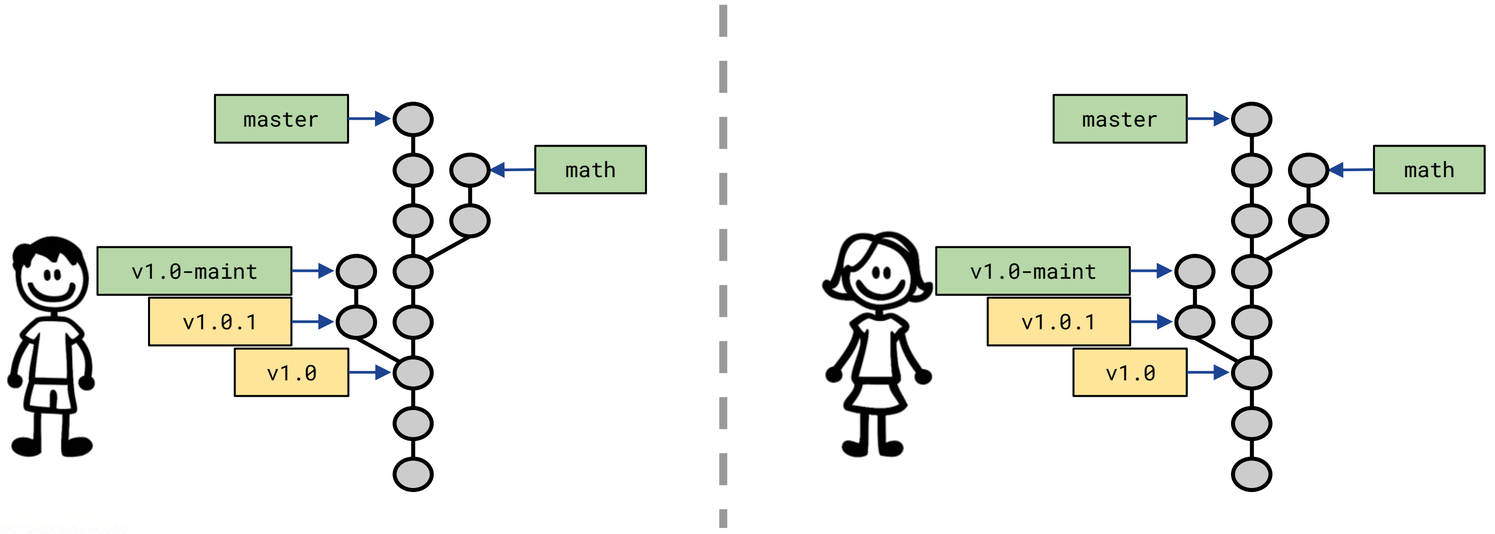


Committed means data is in repository

Staged is modified file marked to go into commit snapshot

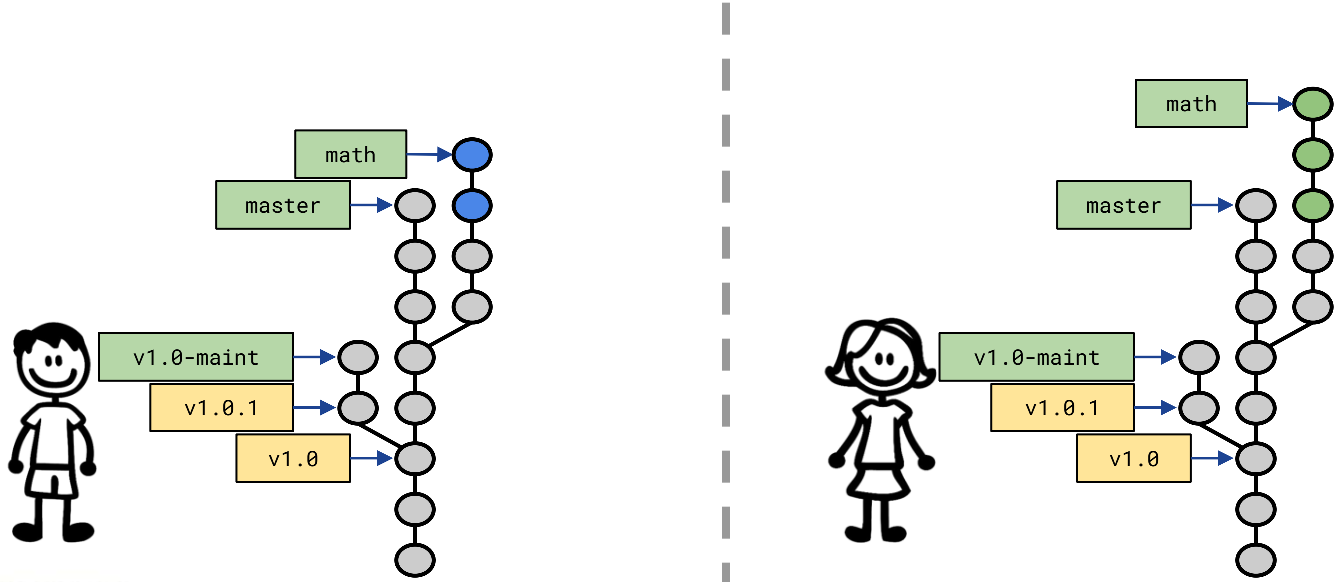
Snapshots and branches

- Bob and Alice start with the same local directory



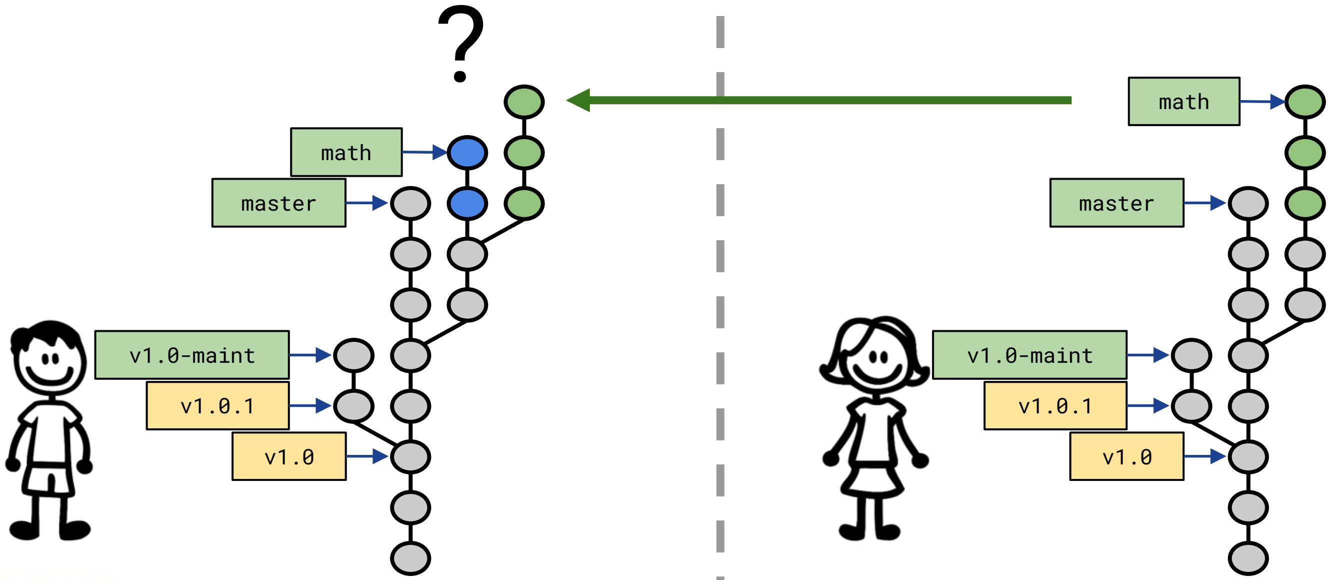
Merges

- Bob and Alice make local changes

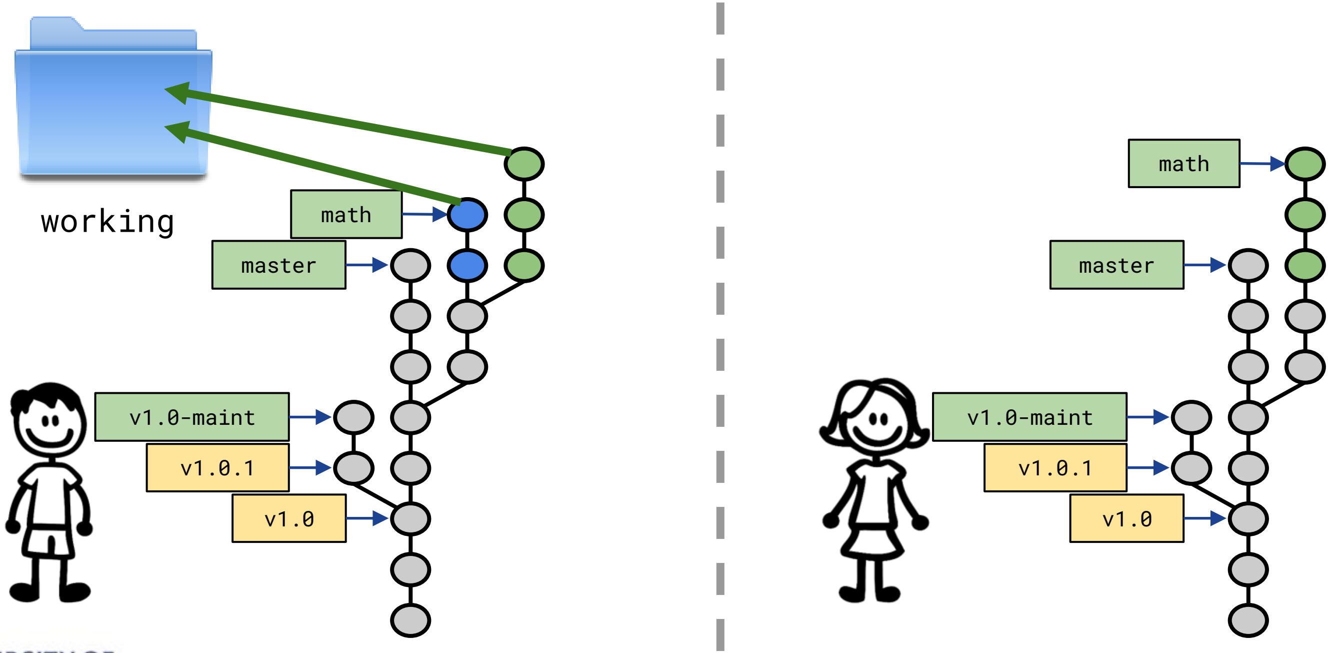


Merges

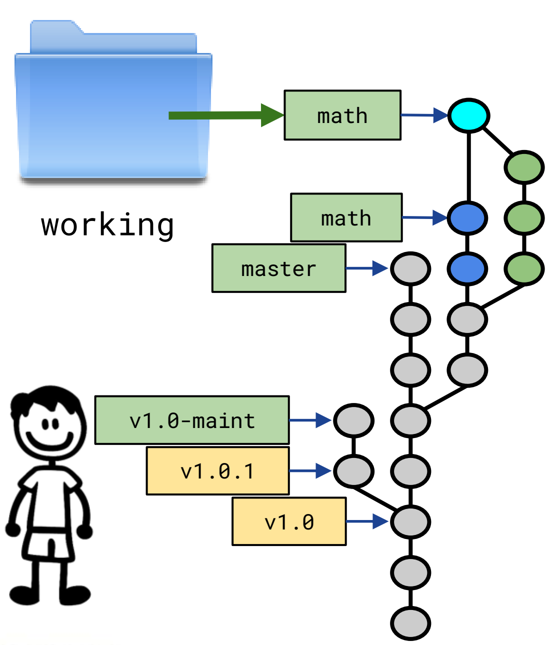
- How to merge the local changes by Bob and Alice?

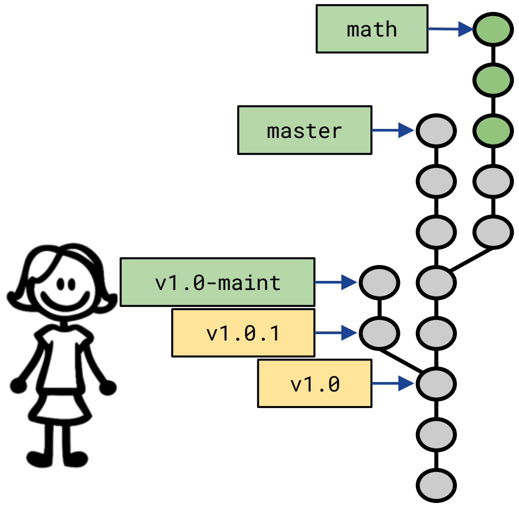


Merges

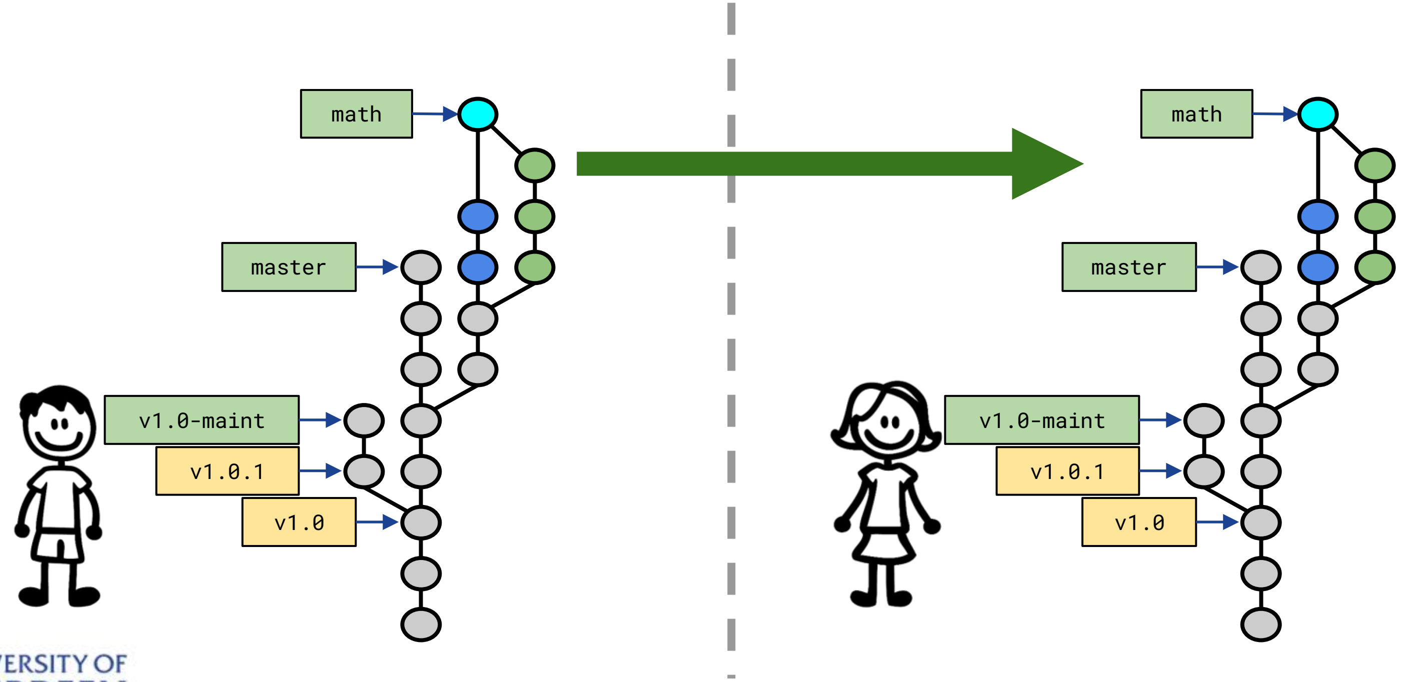

working
math
master
v1.0-maint
v1.0.1
v1.0
math
master
v1.0-maint
v1.0.1
v1.0

Merges

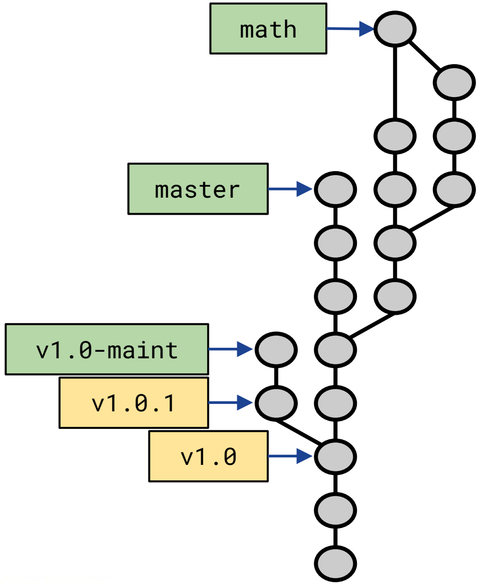

working
math
math
master
v1.0-maint
v1.0.1
v1.0

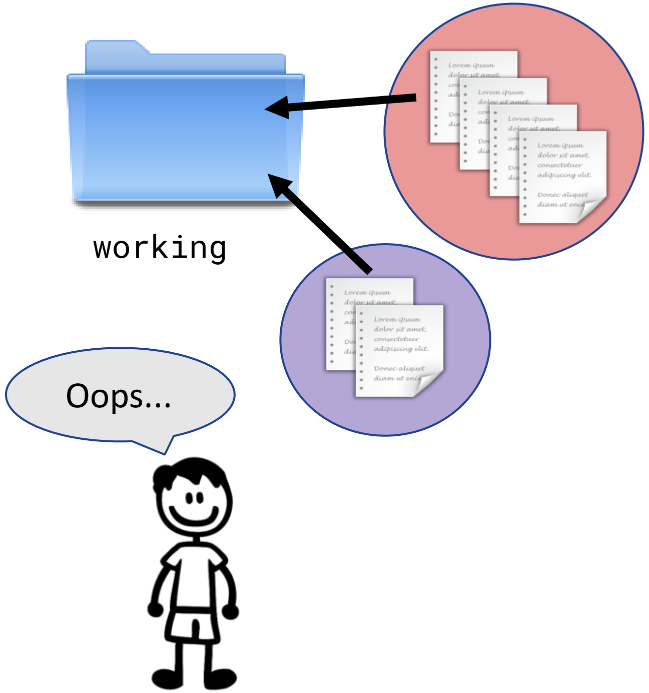

math
master
v1.0-maint
v1.0.1
v1.0

Merges

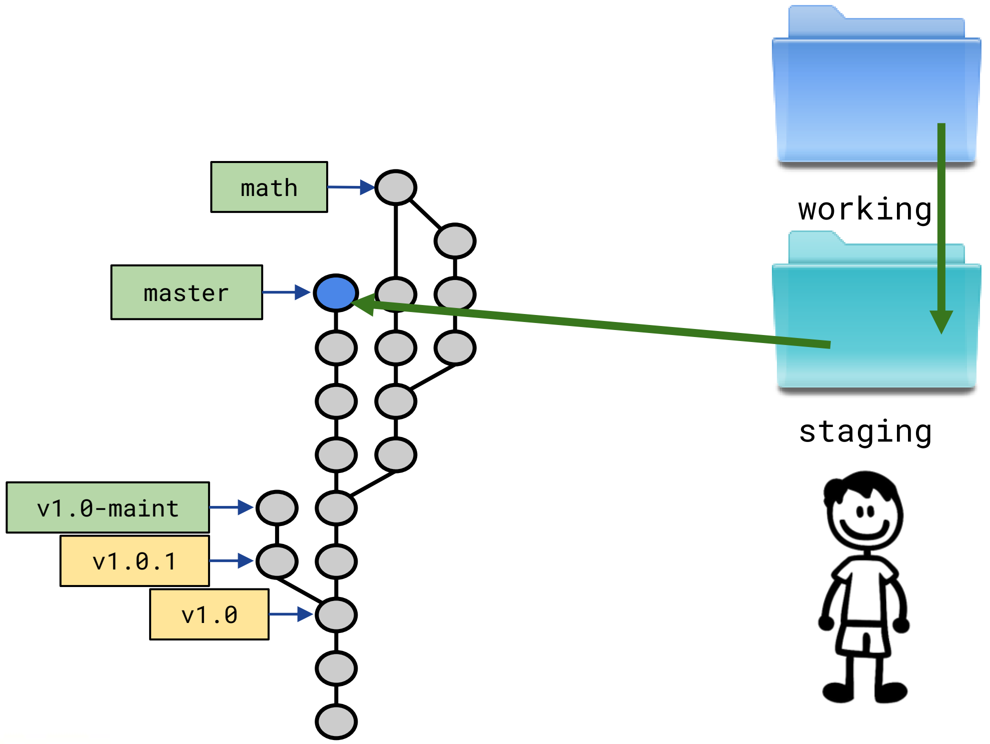

math
master
v1.0-maint
v1.0.1
v1.0
math
master
v1.0-maint
v1.0.1
v1.0

Staging area

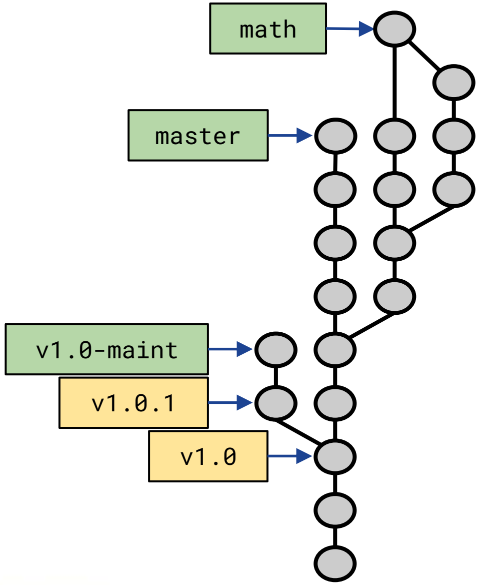


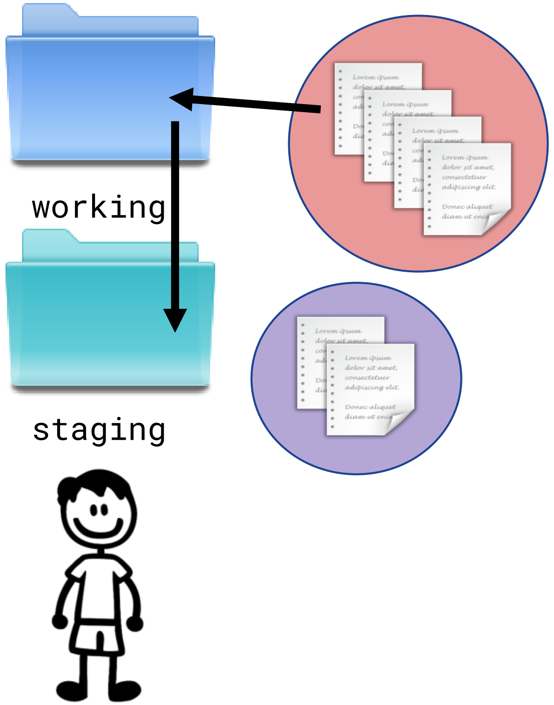


Staging area

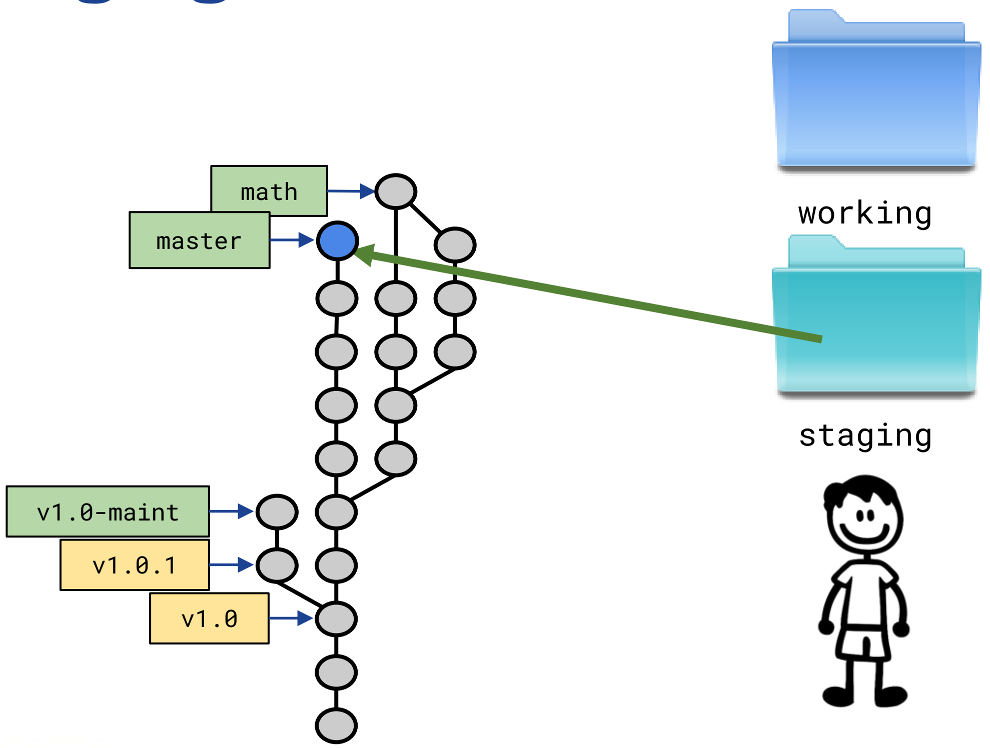


Staging area

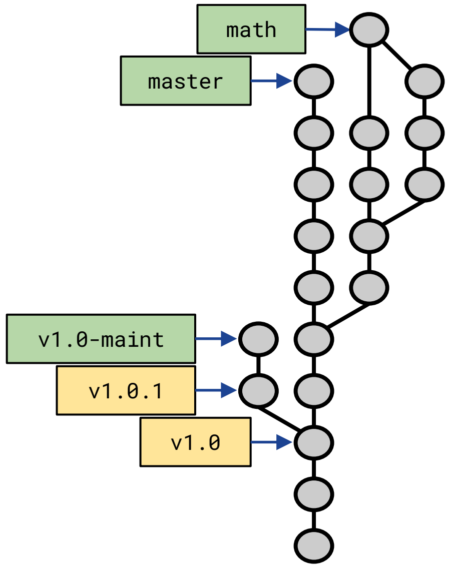


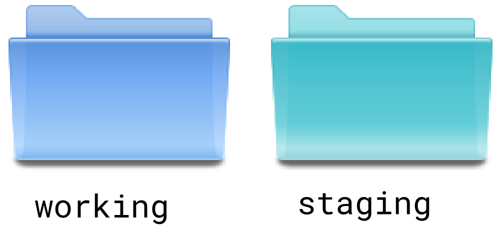


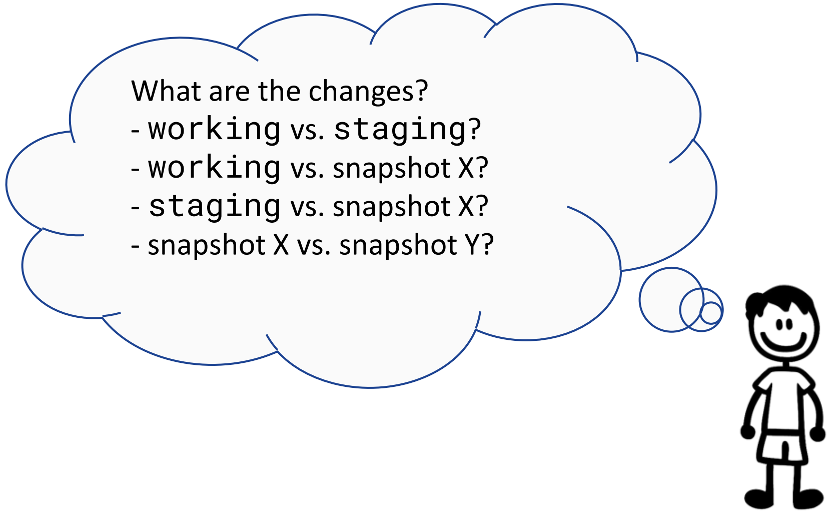
Staging area



Diffs

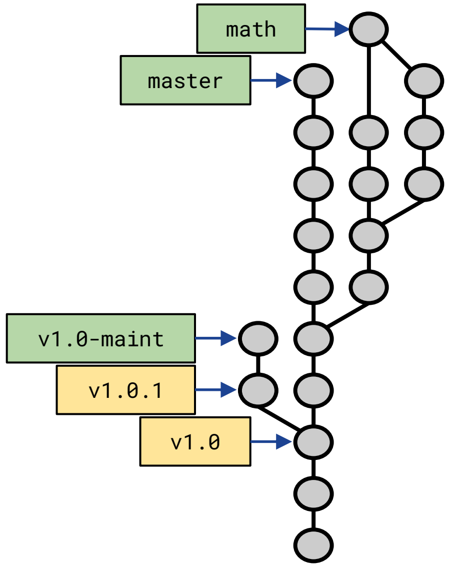


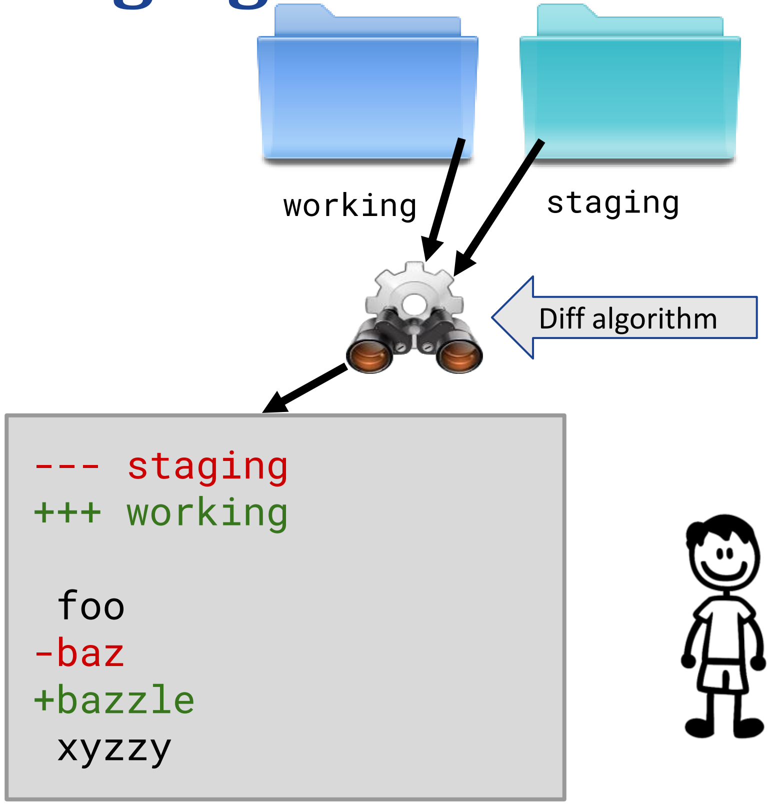




What are the changes?
- `working` vs. `staging`?
- `working` vs. snapshot X?
- `staging` vs. snapshot X?
- snapshot X vs. snapshot Y?

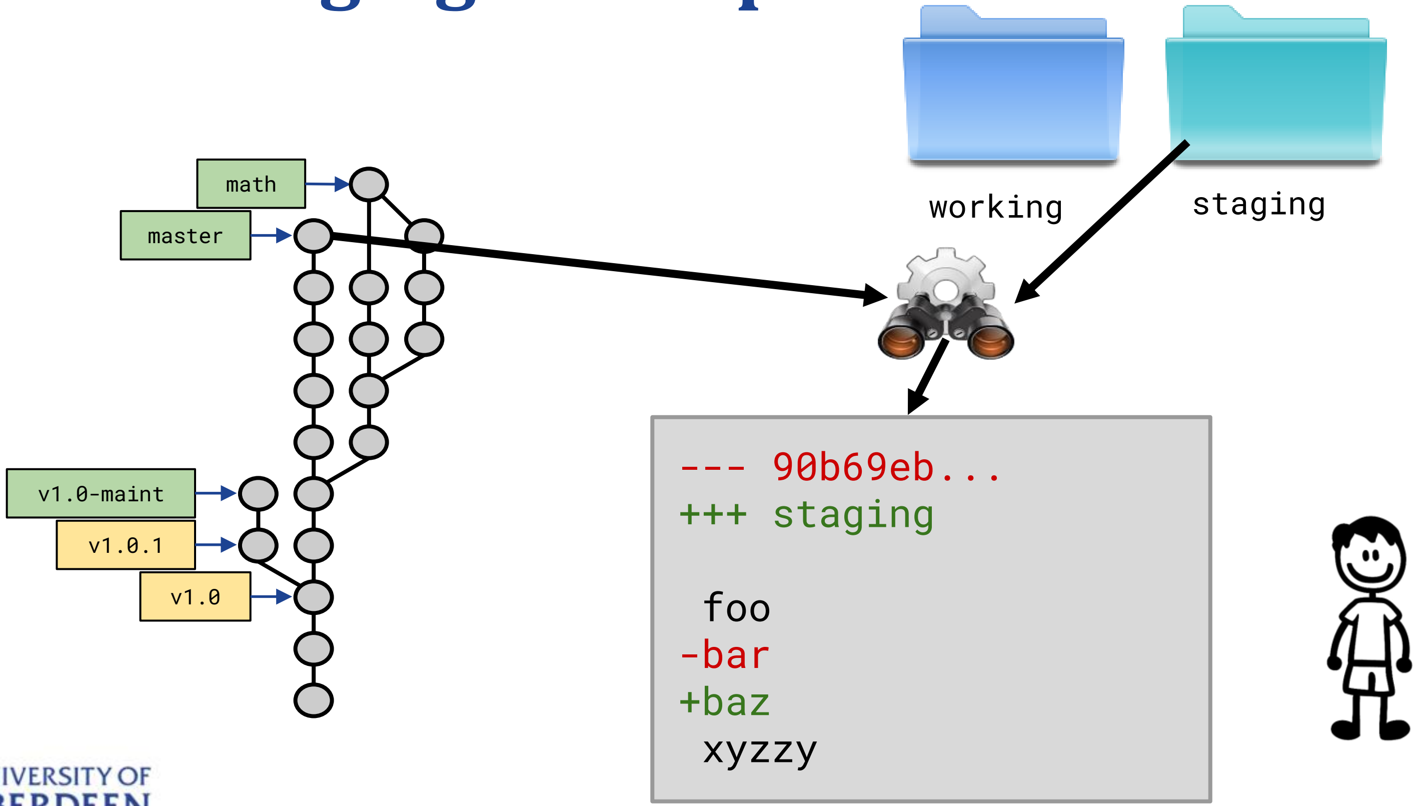
Diffs: working vs. staging





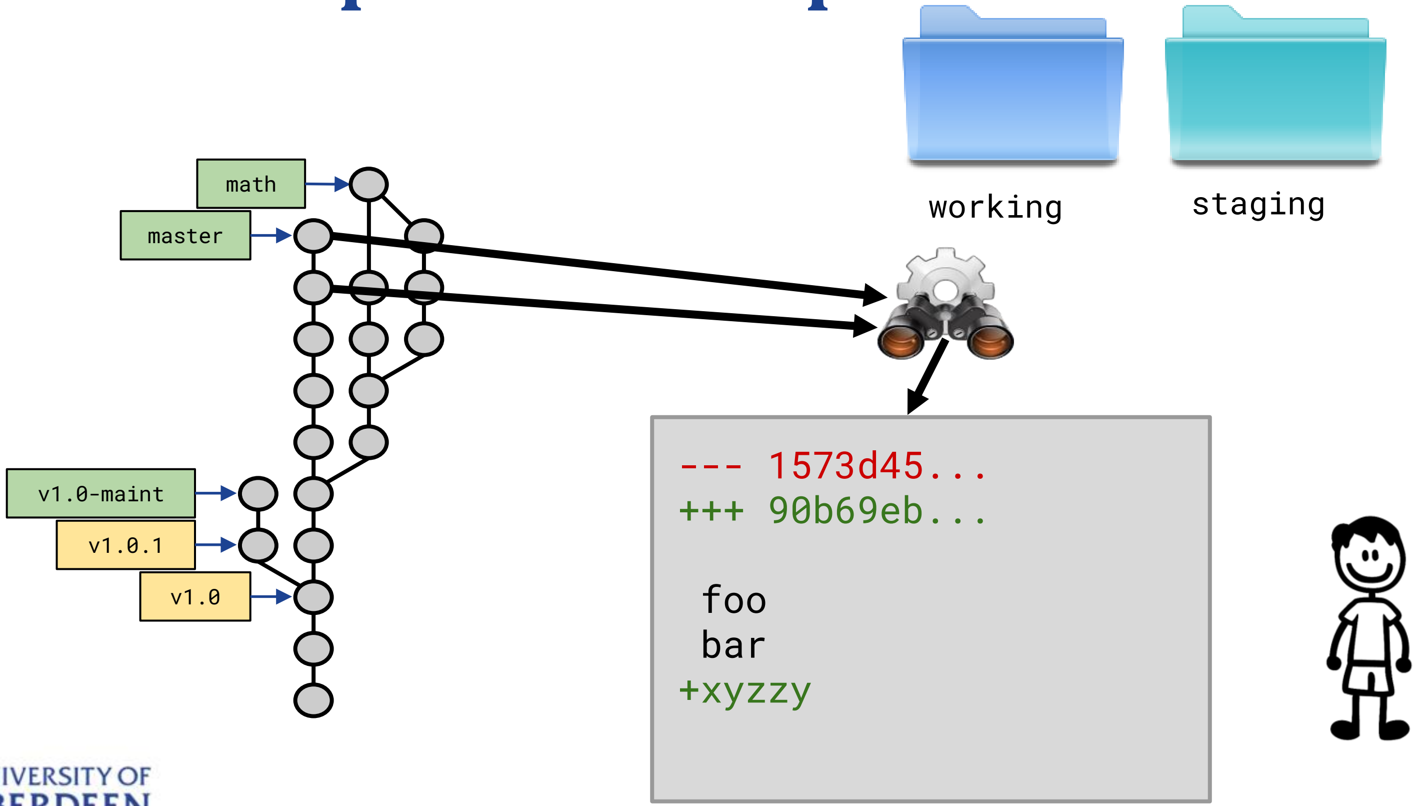
```
--- staging
+++ working

 foo
-baz
+bazzle
 xyzzy
```

Diffs: staging vs. snapshot



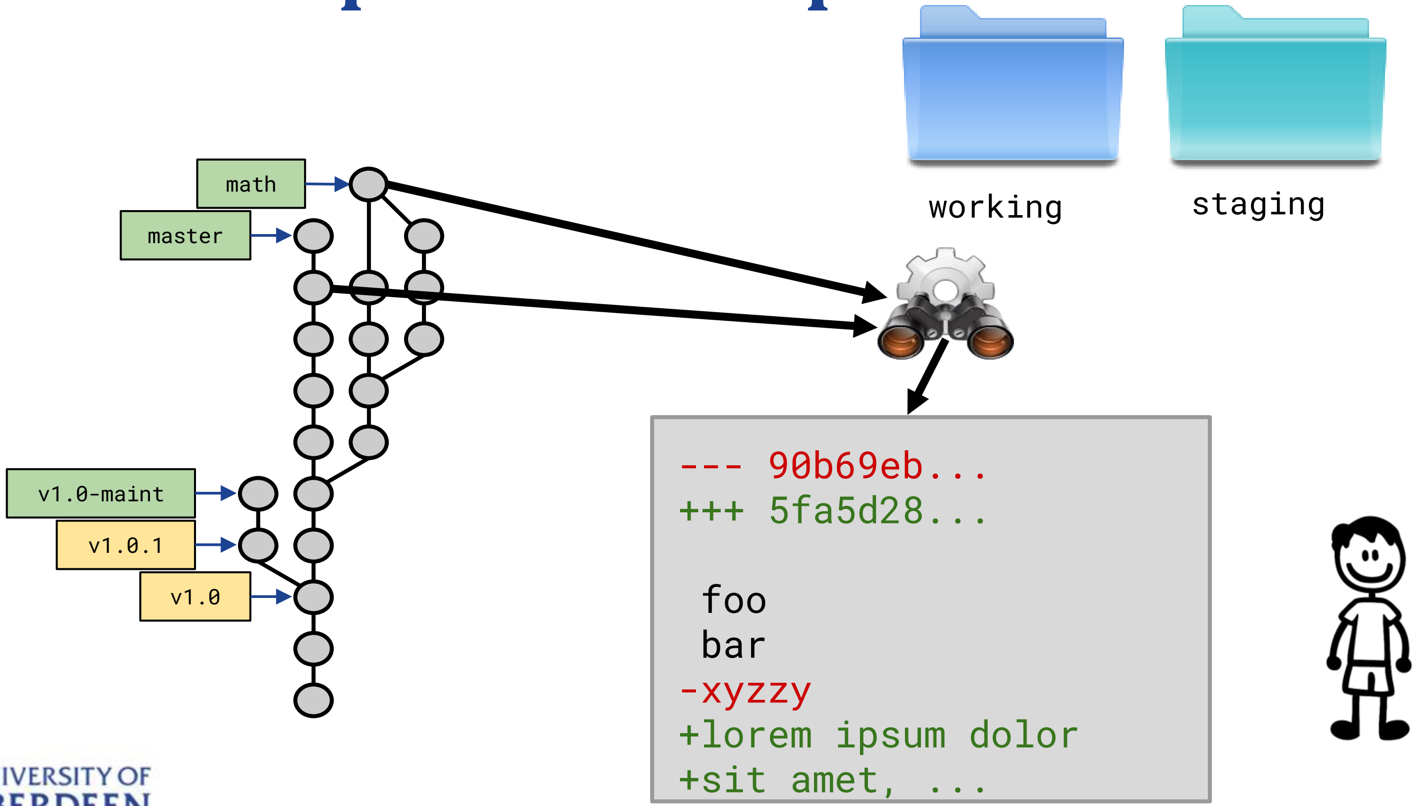
```
--- 90b69eb...
+++ staging

 foo
-bar
+baz
 xyzzy
```

Diffs: snapshot vs. snapshot



```
--- 1573d45...
+++ 90b69eb...

 foo
 bar
+xyzzy
```

Diffs: snapshot vs. snapshot



```
--- 90b69eb...
+++ 5fa5d28...

 foo
 bar
-xyzzy
+lorem ipsum dolor
+sit amet, ...
```

Git commands: getting started

- First, tell Git who you are:

```
git config --global user.name ”My Name”
git config --global user.email ”my@email.address”
```

- Get help:

```
git <command> -h
git help <command>
```

- Start a new Git repository:

```
git init
```

Fetching, merging, pushing

```
git remote add <name> <URL>
```

```
git fetch <name>
git merge <name>/<branch>
```

```
git pull
```

```
git clone <URL> <project>
```

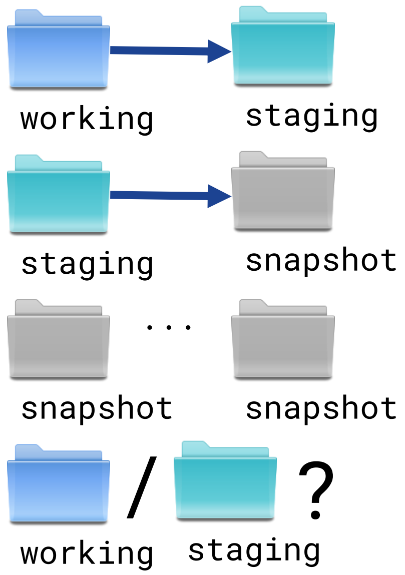
```
git init <project>
cd <project>
git remote add origin <URL>
git fetch origin
git checkout master
```

```
git push origin <name>
```

Adding branches and tags

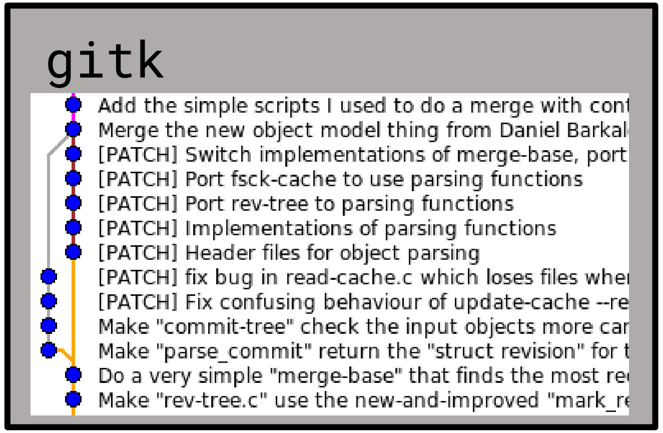
```
git branch

git branch <branch>     }
                        }   git checkout –b <branch>
git checkout <branch>   }

git tag -l

git tag <tag>
```

Making snaphots



```
git add
git commit
git log
git status
```

```
git commit -a
```


gitk

Add the simple scripts I used to do a merge with cont
Merge the new object model thing from Daniel Barkal
[PATCH] Switch implementations of merge-base, port
[PATCH] Port fsck-cache to use parsing functions
[PATCH] Port rev-tree to parsing functions
[PATCH] Implementations of parsing functions
[PATCH] Header files for object parsing
[PATCH] fix bug in read-cache.c which loses files whe
[PATCH] Fix confusing behaviour of update-cache --re
Make "commit-tree" check the input objects more car
Make "parse_commit" return the "struct revision" for t
Do a very simple "merge-base" that finds the most re
Make "rev-tree.c" use the new-and-improved "mark_re

Conflicts

- Merging two branches that modified the same file independently will result in conflict
- If two people work on the same file within the same branch independently the second person's commit will create a conflict
- Conflicts can be resolved manually
- Avoid conflicts by pulling recent changes periodically

```
git pull <branch name>
```

Commands for diffs

working	vs.	staging	`git diff`
staging	vs.	snapshot	`git diff --staged`
working	vs.	snapshot	`git diff HEAD`
snapshot	vs.	snapshot	`git diff <from> <to>`

Further reading about Git

- Oh My Git!: https://ohmygit.org (a game about learning Git)
- Git homepage: http://git-scm.com
- Pro Git: http://git-scm.com/book
- GitHub: http://github.com
- Learn Git Branching: https://learngitbranching.js.org
- Git Ready: http://gitready.com

Questions, comments?

JC2002 Java Programming

Day 1, Session 4: Java language basics

References and learning objectives

- Much of the material is based on slides from ***Java: How to Program***, chapters 2, 4, 5, 7, available via MyAberdeen
- After this session, you should be able to:
  - Write simple Java console programs printing output to the console and receiving user input from the keyboard
  - Use variables, basic arithmetic operations and comparisons

How to write a basic Java program?

- Java programs are organised as *classes* that include *methods*
    - The basic rule is that the source code for a public Java class must be saved in a `.java` file with the same name as the public class
    - Classes and methods will be explained in more detail later
- Every Java program requires one *public* class with *public static* method **`main()`** that serves as starting point of the program
    - When program `MyClass` is started with command **`java MyClass`**, the program starts in method **`main()`** of class **`MyClass`**

Declaring classes and methods

- The basic syntax for declaring a class:

$$\underbrace{\texttt{public static}}_{\text{modifiers (optional)}}\ \underbrace{\texttt{class}}_{\text{keyword }\texttt{class}}\ \underbrace{\texttt{MyClass}}_{\text{class name}}\ \texttt{\{}\ \underbrace{\texttt{…}}_{\text{body of the class (inside curly brackets)}}\ \texttt{\}}$$

- The basic syntax for declaring methods (inside classes):

$$\underbrace{\texttt{protected final}}_{\text{modifiers (optional)}}\ \underbrace{\texttt{void}}_{\text{return type (if empty, use }\texttt{void}\text{)}}\ \underbrace{\texttt{myMethod}}_{\text{method name}}\texttt{(}\underbrace{\texttt{int a, int b}}_{\text{parameters (can be empty)}}\texttt{)}\ \texttt{\{}\ \underbrace{\texttt{…}}_{\text{body of the method (inside curly brackets)}}\ \texttt{\}}$$

Simple Java console program example (1)

```java
// Example text-printing Java program Welcome1.java

public class Welcome1 {
    // main method begins execution of Java application
    public static void main(String[] args) {
        System.out.println("Welcome to Java programming!");
    } // end section main
} // end class Welcome1
```

Simple Java console program example (2)

```java
// Example text-printing Java program Welcome1.java

public class Welcome1 {
    // main method begins execution of Java application
    public static void main(String[] args) {
        System.out.println("Welcome to Java programming!");
    } // end section main
} // end class Welcome1
```

You can add comments that are ignored by the compiler using **//** or **/*** … ***/**

Simple Java console program example (3)

```java
// Example text-printing Java program We

public class Welcome1 {
    // main method begins execution of Java application
    public static void main(String[] args) {
        System.out.println("Welcome to Java programming!");
    } // end section main
} // end class Welcome1
```

Every Java program requires at least one class

Method **`main`** is the starting point of the program

Simple Java console program example (4)

```java
// Example text-printing Java program Welcome1.java

public class Welcome1 {
    // main method begins execution of Java application
    public static void main(String[] args) {
        System.out.println("Welcome to Java programming!");
    } // end section main
} // end class Welcome1
```

Built-in method **`System.out.println`** is used to print one line of text in the console

Simple Java console program example (5)

```java
// Example text-printing Java program Welcome1.java

public class Welcome1 {
    // main method begins execution of Java application
    public static void main(String[] args) {
        System.out.println("Welcome to Java programming!");
    } // end section main
} // end class Welcome1
```

```
$ javac Welcome1.java
$ java Welcome1
Welcome to Java programming!
$
```

Compile the code in file `Welcome1.java`

Run the compiled program `Welcome1`

Output printed by the program

Simple Java console program example 2

```java
// Example text-printing Java program Welcome2.java

public class Welcome2 {
    // main method begins execution of Java application
    public static void main(String[] args) {
        System.out.print("Welcome to ");
        System.out.println("Java programming!");
    } // end section main
} // end class Welcome2
```

```
$ java Welcome2
Welcome to Java programming!
$
```

Simple Java console program example 3

```java
// Example text-printing Java program Welcome3.java

public class Welcome3 {
    // main method begins execution of Java application
    public static void main(String[] args) {
        System.out.println("Welcome\nto\nJava\nprogramming!");
    } // end section main
} // end class Welcome1
```

```
% java Welcome3
Welcome
to
Java
programming!
$
```

Escape sequences

- You can format console output using *escape sequences*

Escape sequence	Description
\n	Newline. Position the screen cursor at the beginning of the *next* line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the cursor at the beginning of the current line. Any characters output after \r will overwrite the characters on the current line.
\\	Backslash. Used to print backslash character (\\).
\\”	Double quote. Used to print double quote character (“).

Format output with printf

- Use `System.out.printf` method for printing output to the console
  - "`f`" means "formatted": `printf` displays *formatted* data
- The arguments are placed in a *comma-separated list*
- Calling a method is also referred to as *invoking* a method
- Java allows large statements to be split over many lines
  - However, cannot split a statement in the middle of an identifier or string

Arguments for printf

- Method `printf`’s first argument is a *format string*
  - May consist of *fixed text* and *format specifiers*
  - Fixed text is output as it would be by `print` or `println` method
  - Each format specifier is a placeholder for a value and specifies the type of data to output
- Format specifiers are a percent sign (%) followed by a character that represents the data type
- For a string, **`%s`** is a placeholder, for an **`int`**, **`%d`** is a placeholder
- Other placeholders: **`%f`** for floating point, **`%b`** for **`boolean`**

Formatted printing example

```java
// Example formatted printing Java program

public class Welcome4 {
    // main method begins execution of Java application
    public static void main(String[] args) {
        System.out.printf("%s%n%s%n", "Welcome to", "Java programming!");
    } // end section main
} // end class Welcome4
```

```
Welcome to
Java programming!
```

- Note that **\n** and **%n** both can be used for newline

Declaring variables in Java

- In Java, variables need to be declared before using them
- Basic syntax for declaring a variable with default value:

$$\underbrace{\texttt{public}}_{\text{modifiers (optional)}}\ \underbrace{\texttt{int}}_{\text{data type}}\ \underbrace{\texttt{number}}_{\text{variable name}}\texttt{;}$$

- Basic syntax of declaring a variable and assigning it a value:

$$\underbrace{\texttt{private}}_{\text{modifiers (optional)}}\ \underbrace{\texttt{double}}_{\text{data type}}\ \underbrace{\texttt{number}}_{\text{variable name}}\ \texttt{=}\ \underbrace{\texttt{5.8}}_{\text{assigned value}}\texttt{;}$$

1495 UNIVERSITY OF ABERDEEN

Primitive types in Java

- Primitive types are not derived from other data types

Type	Size (bits)	Value range
`boolean`	1	`true` or `false`
`char`	16	0 to 65535
`byte`	8	-128 to +127 (-2^7 to $+2^7-1$)
`short`	16	-32,768 to +32,767 (-2^{15} to $+2^{15}-1$)
`int`	32	-2^{31} to $+2^{31}-1$ (about $-2\cdot10^9$ to $+2\cdot10^9$)
`long`	64	-2^{64} to $+2^{64}-1$ (about -10^{19} to $+10^{19}$)
`float`	32	about (+/-) $1.4\cdot10^{-45}$ to $3.4\cdot10^{38}$
`double`	64	about (+/-) $4.9\cdot10^{-324}$ to $1.8\cdot10^{308}$

1495 UNIVERSITY OF ABERDEEN

Formatted printing with input and integers

```java
// Example formatted printing Java program with input
import java.util.Scanner; // needed for input
public class Addition {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter first integer: "); // prompt
        int number1 = input.nextInt(); // read first number
        System.out.print("Enter second integer: "); // prompt
        int number2 = input.nextInt(); // read first number
        int sum = number1 + number2; // add numbers and store the result
        System.out.printf("Sum is: %d%n", sum);
    } // end section main
} // end class Addition
```

```
Enter first integer: 42
Enter second integer: 88
Sum is: 130
```

Imported classes

- By default, package **`java.lang`** is imported in every Java program; thus, classes in **`java.lang`** (such as **`System`**) are the only ones that don't need to be imported
- In the previous example, **`Scanner`** class is needed to enable a program to read data for use in a program
  - Data can come from many sources, such as the user at the keyboard or a file on disk
  - Before using a **`Scanner`**, you must create it and specify the source of the data

Binary overflow

- In some other programming languages (like C), variables may overflow their range of allocated bits
  - For example, for bytes, 127+1 results -128
- In Java, such situations are usually prevented and will give an error, but binary overflows can still occur for `int` and `long`. Be cautious with large numbers!

```
$ java Addition
Enter first integer: 2000000000
Enter second integer: 2000000000
Sum is: -294967296
```

Formatted printing of floats

```java
// Example formatted printing Java program with input
import java.util.Scanner; // needed for input
public class Addition2 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter x: "); // prompt
        float x = input.nextFloat(); // read first number
        System.out.print("Enter y: "); // prompt
        float y = input.nextFloat(); // read first number
        float sum = x + y; // add numbers and store the result
        System.out.printf("Sum is: %f%n", sum);
        System.out.printf("Sum is: %.2f%n", sum);
    } // end section main
} // end class Addition2
```

```
Enter x: 1.255
Enter y: 2.75
Sum is: 4.005000
Sum is: 4.01
```

Arithmetic operations in Java

Java operation	Operator	Algebraic expression	Java expression
Addition	+	$f + 78$	`f + 78`
Subtraction	-	$f - c$	`f – c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

UNIVERSITY OF ABERDEEN

Arithmetic precedence in Java

- Arithmetic operations in Java follow the standard precedence
    - **Multiplication**, **division**, and **remainder** (*, / ,%) are evaluated first. If there are several operators of this type, they are evaluated from *left* to *right*
    - **Addition** and **subtraction** (+, -) are evaluated next. Multiple operators of this type are evaluated from *left* to *right*.
    - **Assignment** (=) is evaluated last
- To improve readability and to avoid mistakes, you can use parenthesis in complex expressions.

Java equality and relational operators

Algebraic operator	Java operator	Example Java condition	Meaning
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Comparisons and booleans example

```java
// Example comparison Java program
import java.util.Scanner; // needed for input
public class Comparison {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter x: "); // prompt
        int x = input.nextInt(); // read first number
        System.out.print("Enter y: "); // prompt
        int y = input.nextInt(); // read first number
        boolean isLarger = x > y; // compare if x is larger than y
        boolean isLess = x < y; // compare if x is less than y
        System.out.printf("x is larger than y: %b%n", isLarger);
        System.out.printf("x is less than y: %b%n", isLess);
    } // end section main
} // end class Comparison
```

```
Enter x: 5
Enter y: 10
x is larger than y: false
x is less than y: true
```

```
Enter x: 5
Enter y: 5
x is larger than y: false
x is less than y: false
```

```
Enter x: 10
Enter y: 5
x is larger than y: true
x is less than y: false
```

Summary

- Java is an old language widely used on many platforms
    - Java bytecodes are portable without re-compilation
- Java development cycle involves five phases
    - Writing and editing the code, compiling to bytecode, loading the bytecode, verifying the bytecode, and executing the program
- Version control is essential for program development in collaboration with others
    - Allows merging changes by different persons as well as rolling back to earlier versions of the software
- Basic syntax and structure of Java programs introduced

Questions, comments?

1495 UNIVERSITY OF ABERDEEN