

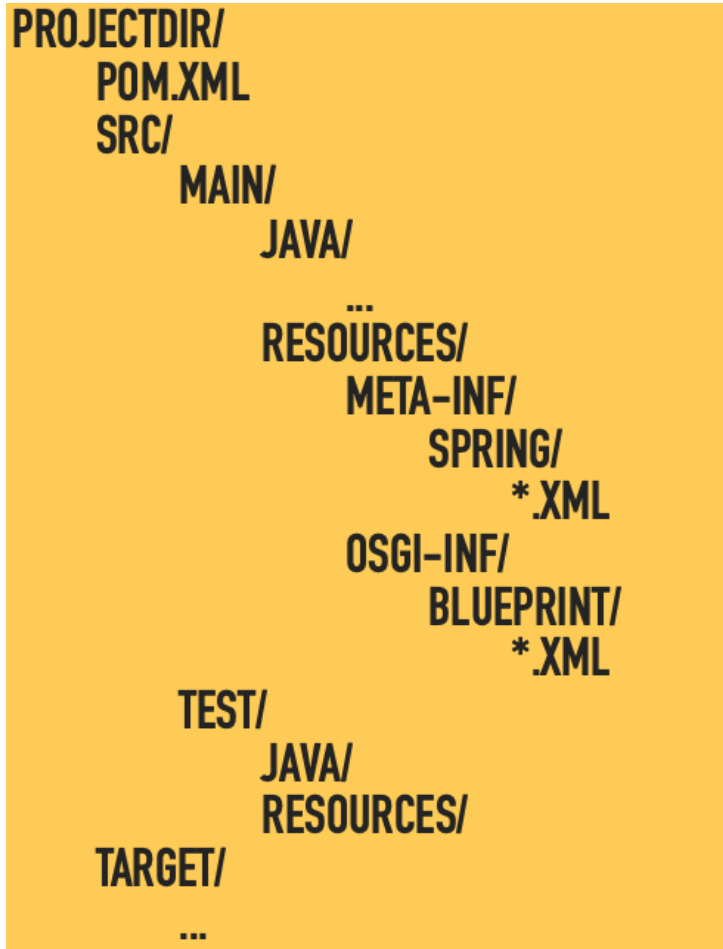
Maven

What is Maven?

- ❖ A **project management tool**, which encompasses
 - a project object model,
 - a set of standards,
 - a project lifecycle,
 - a dependency management system and
 - logic for executing plugin goals at defined phases in a lifecycle.
- ❖ Convention over Configuration
 - (standardised project layout)
- ❖ Example of use

```
$ mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false
```

Standardised folder layout



- ❖ **POM** contains a complete description of how to build the project
- ❖ **src** directory contains all of the source material for building the project, its site and so on.
- ❖ **target** directory contains the results of the build (typically a JAR), as well as all the intermediate files.

The POM

- ❖ Maven is based on the concept of
 - **Project Object Model (POM)**
- ❖ XML file, always residing in the base directory of the project as **pom.xml**. *→ estrutura para o maven*
 - Users defined POMs extend the Super POM.
- ❖ The POM contains information about the project and various configuration detail used by Maven to build the project.
- ❖ The Maven POM is declarative
 - No procedural details are needed.

POM Structure

❖ The POM contains four categories of description and configuration:

1. General project information

- human-readable information

2. Build Settings

- add plugins, attach plugin goal to lifecycle

3. Build Environment

- describe the “family” environment in which Maven lives

❖ POM Relationships

- coordinates, inheritance, aggregations, dependencies

An Example of POM

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>pt.ua.deti.ies</groupId>
6   <artifactId>WeatherRadar</artifactId>
7   <version>0.0.1-SNAPSHOT</version>
8   <packaging>jar</packaging>
9   <name>WeatherRadar</name>
10  <url>http://maven.apache.org</url>
11
12  <licenses>
13    <license>
14      <name>Apache 2</name>
15      <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
16      <distribution>repo</distribution>
17      <comments>A business-friendly OSS license</comments>
18    </license>
19  </licenses>
20
21  <organization>
22    <name>UA_IES</name>
23    <url>http://www.ua.pt</url>
24  </organization>
```

Annotations:

- sob directórios* (above directories) pointing to `pt.ua.deti.ies`
- GroupID é o "package" ☺* (GroupID is the "package" ☺) pointing to `pt.ua.deti.ies`
- primeiro diretório* (first directory) pointing to `WeatherRadar`
- Incrementa o dia e a hora!* (Increment the day and the hour!) pointing to `0.0.1-SNAPSHOT`
- IMPORTANT** pointing to `<packaging>jar</packaging>`

Run:

```
$ mvn compile
$ mvn package
```

An Example of POM

```
43 <properties>
44   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
45   <maven.compiler.source>14</maven.compiler.source>
46   <maven.compiler.target>14</maven.compiler.target>
47 </properties>
48
49 <dependencies>
50   <dependency>
51     <groupId>junit</groupId>
52     <artifactId>junit</artifactId>
53     <version>3.8.1</version>
54     <scope>test</scope>
55   </dependency>
56   <dependency>
57     <groupId>com.squareup.retrofit2</groupId>
58     <artifactId>retrofit</artifactId>
59     <version>2.6.1</version>
60   </dependency>
61   <dependency>
62     <groupId>com.squareup.retrofit2</groupId>
63     <artifactId>converter-gson</artifactId>
64     <version>2.6.1</version>
65   </dependency>
66   <dependency>
67     <groupId>log4j</groupId>
68     <artifactId>log4j</artifactId>
69     <version>1.2.12</version>
70     <scope>compile</scope>
71   </dependency>
72 </dependencies>
--
```

lista as dependencias

An Example of POM

```
74 <build>
75   <plugins>
76
77     <plugin>
78       <groupId>org.apache.maven.plugins</groupId>
79       <artifactId>maven-site-plugin</artifactId>
80       <version>3.8.2</version>
81     </plugin>
82
83     <plugin>
84       <groupId>org.apache.maven.plugins</groupId>
85       <artifactId>maven-project-info-reports-plugin</artifactId>
```


Maven Coordinates

- ❖ Coordinates define the unique place of the project in the Maven universe.
- ❖ They are made up of <groupId>, <artifactID> and <version> (The Maven trinity!) 📌
- ❖ Project versions are used to group and order releases:
 - <major version>.<minor version>.<incremental version>-<qualifier>
 - E.g.: 1.2.3-alpha-2
major minor incremental qualifier
- ❖ If the qualifier contains the keyword –SNAPSHOT
 - Maven will expand this token to a date and time value converted to UTC.

Maven Coordinates

❖ **groupId**

- Name of the company, organization, team etc., usually using the reverse URL naming convention

❖ **artifactId**

- A unique name for the project under groupId

❖ **version**

❖ **packaging**, default: jar

❖ **classifier**

Maven coordinates uniquely identifies a project.

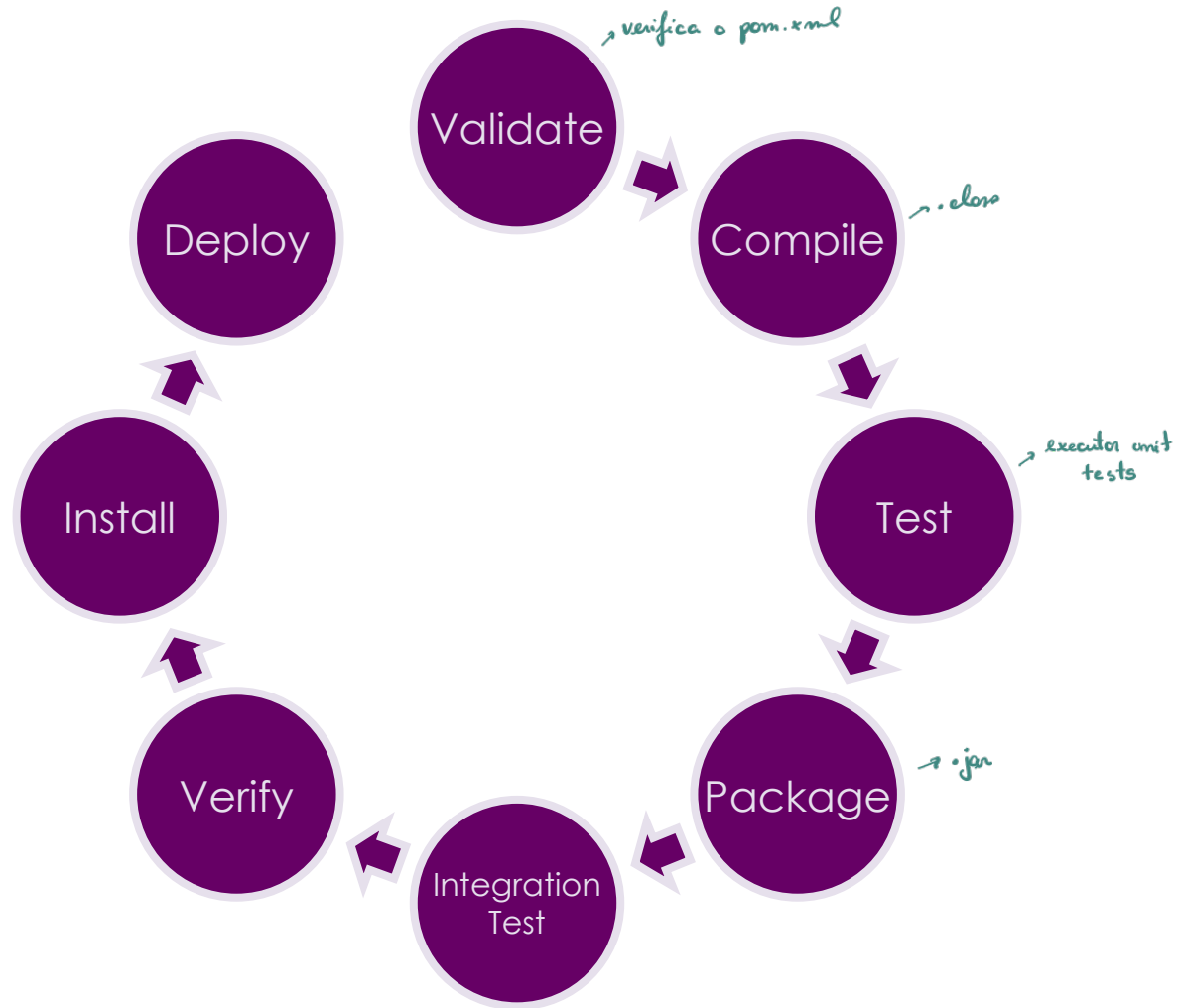
Maven Lifecycle

- ❖ A lifecycle is an organised sequence of phases that give order to a sequence of goals.
- ❖ Goals are packaged in plugins that are tied to phases.
- ❖ Calling a specific phase in a build cycle will trigger every prior build phase.

Clean Lifecycle
pre-clean
clean
post-clean

Default Lifecycle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	package
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	install
generate-test-resources	deploy
process-test-resources	

Maven Lifecycle



Maven Lifecycle

1. Validate

- Validates the project structure is correct.
 - For example – It checks if all the dependencies have been downloaded and are available in the local repository.

2. Compile

- It compiles the source code, converts the .java files to .class, and stores the classes in the target/classes folder.

3. Test

- It runs unit tests for the project.

4. Package

- It packages the compiled code in a distributable format like JAR or WAR.

Maven Lifecycle

5. Integration test

- It runs the integration tests for the project.

6. Verify

- It runs checks to verify that the project is valid and meets the quality standards.

7. Install

- It installs the packaged code to the local Maven repository.

8. Deploy

- It copies the packaged code to the remote repository for sharing it with other developers.

Build Lifecycle

- ❖ The process for building and distributing a project
- ❖ A build lifecycle consists of several steps called phases
- ❖ Some Default Lifecycle Phases
 - validate
 - compile
 - test
 - package
 - deploy

Goals and Plugins

- ❖ Goals are operations provided by Maven plugins
- ❖ Each phase is a sequence of goals
 - Each goal is responsible for a specific task
- ❖ When we run a phase, all goals bound to this phase are executed in order
- ❖ Some Maven Plugins
 - resources
 - compiler
 - surefire
 - jar, war

Archetype

Templates são bons ! //

- ❖ An archetype is a *template* for a Maven project which can be used to create new projects quickly
- ❖ Example: creating a project from archetype
 - `maven-archetype-quickstart`
 - `maven-archetype-webapp`

} Estrutura pré-feita
- ❖ Users can create new archetypes and publish them through catalogs
 - Main Maven archetype catalog:
<http://repo1.maven.org/maven2/archetype-catalog.xml>

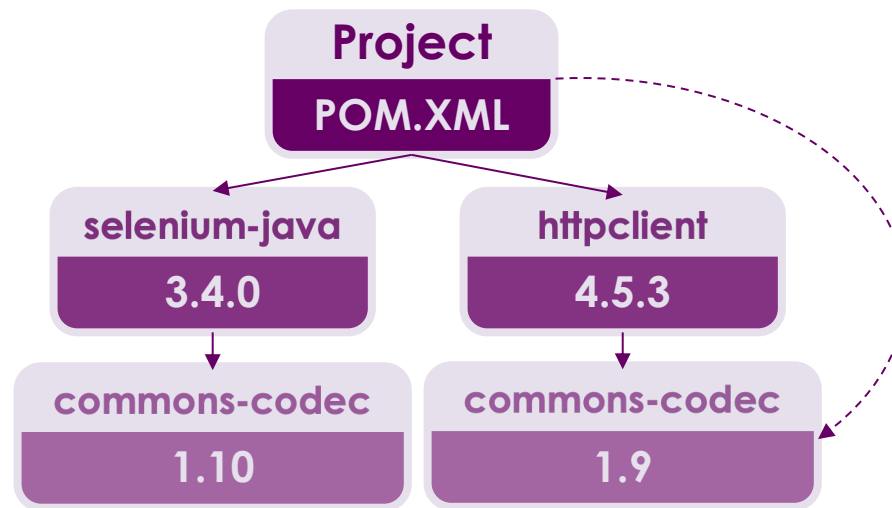
Dependency Management

- ❖ A dependency of a project is a library that the project depends on
- ❖ Adding a dependency to a project is as simple as adding the coordinates of the library to `pom.xml`
- ❖ Maven *automatically downloads the library from an online repository* and store it locally for future use



Dependencies and Repositories

- ❖ Search for dependency coordinates
 - <http://mvnrepository.com/>
 - <https://search.maven.org>
- ❖ Maven Central Repository
 - <https://repo.maven.apache.org/maven2/>



Git and GitHub

Why version control?

❖ Scenario 1

- Your program is working
- You change “just one thing”
- Your program breaks
- You change it back
- Your program is still broken--why?

❖ Scenario 2

- Your program worked well enough yesterday
- You made a lot of improvements last night...
 - ...but you haven't gotten them to work yet
- You need to turn in your program now

Version control for teams

❖ Scenario 1

- You change one part of a program--it works
- Your co-worker changes another part--it works
- You put them together--it doesn't work
- Some change in one part must have broken something in the other part
- What were all the changes?

❖ Scenario 2

- You make a number of improvements to a class
- Your co-worker makes a number of different improvements to the same class
- How can you merge these changes?

Version control systems

- ❖ A version control system (often called a source code control system) does these things:
 - Keeps multiple (older and newer) versions of everything (not just source code)
 - Requests comments regarding every change
 - Allows “check in” and “check out” of files so you know which files someone else is working on
 - Displays differences between versions

Benefits of version control

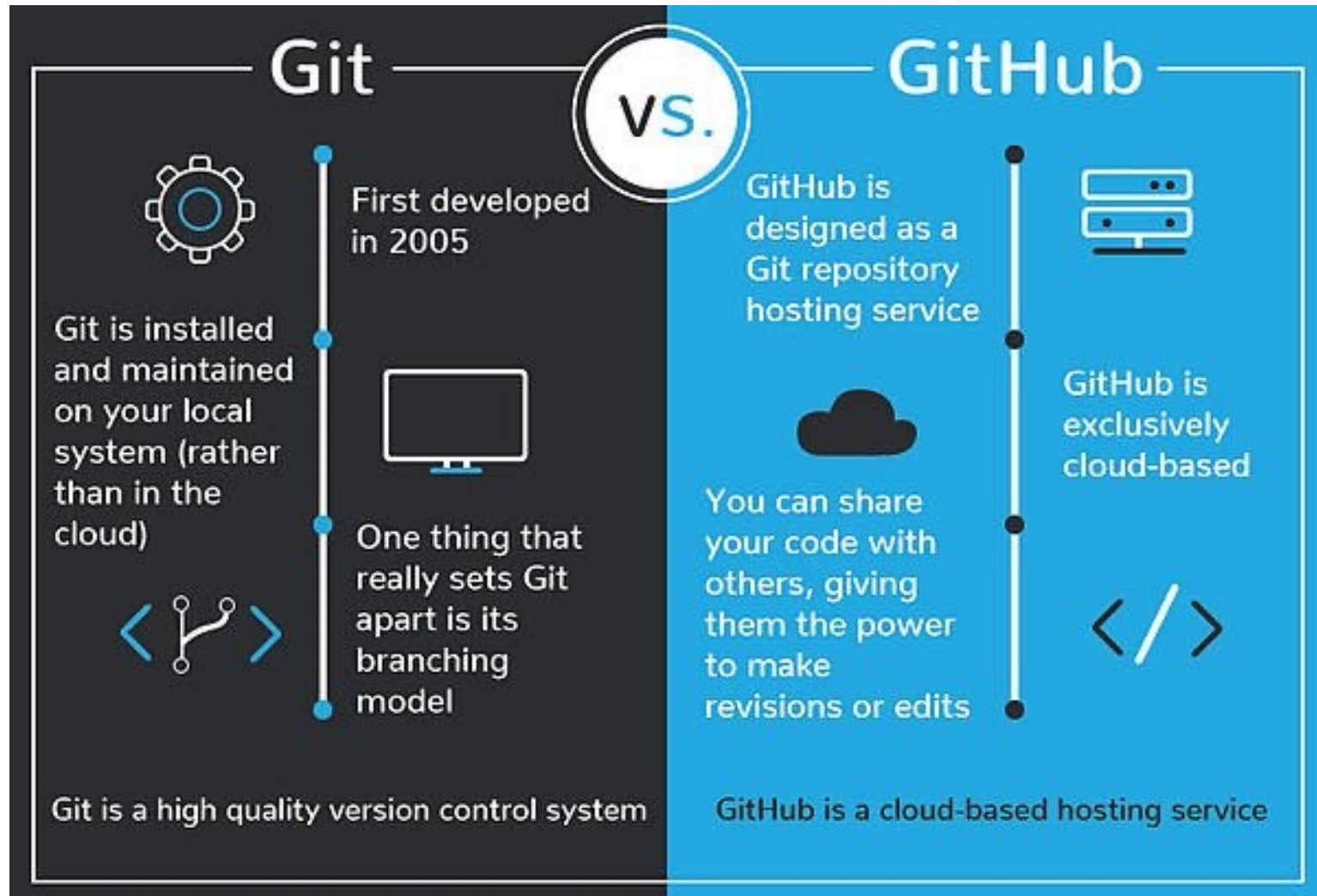
❖ For working by yourself:

- Gives you a “time machine” for going back to earlier versions
- Gives you great support for different versions (standalone, web app, etc.) of the same basic project

❖ For working with others:

- Greatly simplifies concurrent work, merging changes

What are Git and GitHub



Introduce yourself to Git

- ❖ On your computer, open the Git Shell application.
- ❖ Enter these lines (with appropriate changes):

```
git config --global user.name "John Smith"  
git config --global user.email jsmith@seas.upenn.edu
```
- ❖ You only need to do this once
- ❖ If you want to use a different name/email address for a particular project, you can change it for just that project
 - cd to the project directory
 - Use the above commands, but leave out the `--global`

init and the .git repository

- ❖ When you execute the command **git init** in your project directory, or when you clone an existing project, you create a repository
 - The repository is a subdirectory named `.git` containing various files
 - The dot indicates a “hidden” directory
 - You do not work directly with the contents of that directory; various git commands do that for you
- ❖ At any time, you can take a “snapshot” of everything (or selected things) in your project directory, and put it in your repository
 - This “snapshot” is called a **commit object**

Making commits

- ❖ You do your work in your project directory, as usual
- ❖ If you create new files and/or folders, they are not tracked by Git unless you ask it to do so
`git add newFile1 newFolder1 newFolder2 newFile2`
- ❖ Committing makes a “snapshot” of everything being tracked into your repository
 - A message telling what you have done is required
`git commit -m “Uncrevulated the conundrum bar”`
`git commit`

Commits and graphs

- ❖ A **commit** is when you tell git that a change (or addition) you have made is ready to be included in the project
- ❖ When you commit your change to git, it creates a **commit object**
 - a complete state of the project, including all the files in the project
 - The very first commit object has no “parents”
 - Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object
 - Hence, most commit objects have a single parent
 - You can also **merge** two commit objects to form a new one
 - The new commit object has two parents
- ❖ Hence, commit objects forms a directed graph
 - **Git is all about using and manipulating this graph**

Commit messages

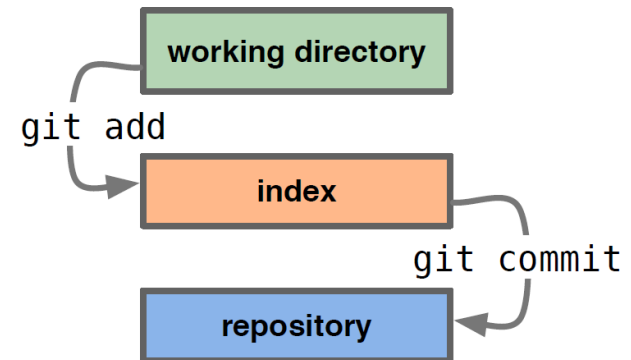
- ❖ In git, “Commits are cheap.” Do them often.
- ❖ When you commit, you must provide a one-line message stating what you have done
 - Terrible message: “Fixed a bunch of things”
 - Better message: “Corrected the calculation of median scores”
- ❖ Commit messages can be very helpful, to yourself as well as to your team members
- ❖ You can't say much in one line, so commit often

Typical workflow

`git status`

- See what Git thinks is going on
- Use this frequently!

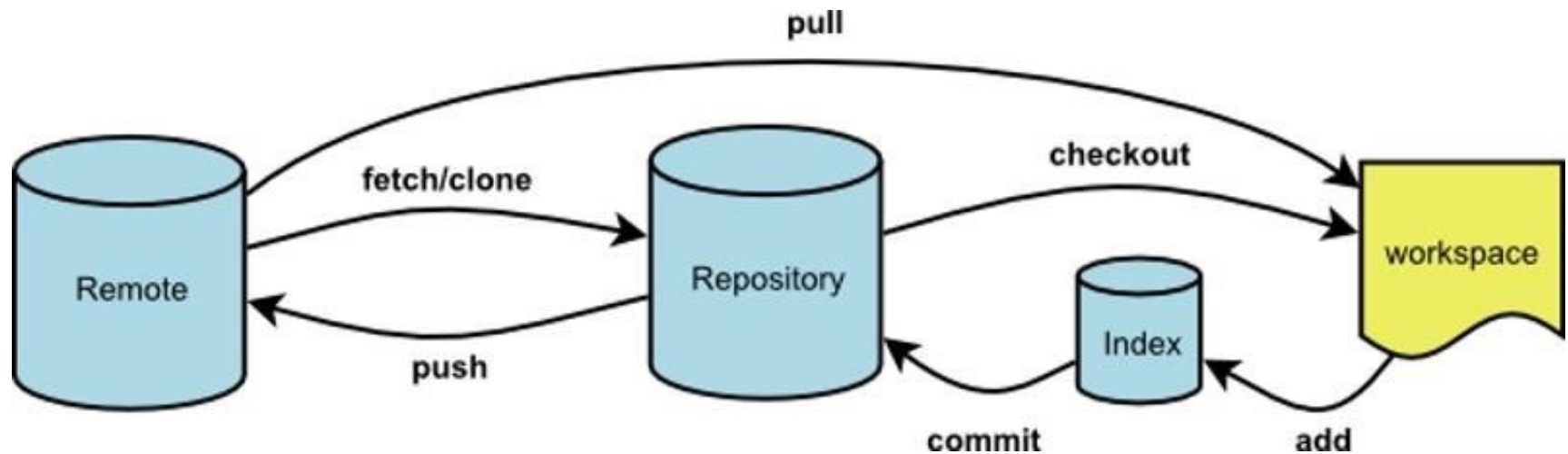
❖ Work on your files
`git add your editfiles`
`git commit -m "What I did"`



Keeping it simple

- ❖ If you:
 - Make sure you are current with the central repository
 - Make some improvements to your code
 - Update the central repository before anyone else does
- ❖ Then you don't have to worry about resolving conflicts or working with multiple branches
 - All the complexity in git comes from dealing with these
- ❖ Therefore:
 - Make sure you are up-to-date before starting to work
 - Commit and update the central repository frequently
- ❖ If you need help: <https://help.github.com/>

More Commands



Remote repositories

- ❖ GitHub
- ❖ GitLab
- ❖ Bitbucket
- ❖ Amazon AWS CodeCommit
- ❖ Codebase
- ❖ Microsoft Azure DevOps
- ❖ SourceForge

Introduce yourself to GitHub

- ❖ Register on GitHub
 - <https://github.com/>
- ❖ Authenticating to GitHub Desktop
- ❖ Create or add a repository to GitHub
- ❖ Commit your changes on GitHub
- ❖ Creating a branch for your work
- ❖ Synchronizing your branch