

Introduction to Tools

Module 1: 01

About the Instructor: Dr. Isaac Chow



Doctor of Engineering
2010-2017



BS -1984
MS 1985



1985 -1989
Computer Science
Instructor



1989 -1992
Programmer



1992 -2000
Senior Technical
Staff Member



2001 -2003
Computer Scientist



2005 - present
Senior Software Engineer



About the Instructor

- **Title:** Adjunct Professor
 - **Company:** Southern Methodist University
 - **Location:** Dallas/Fort Worth Area, TX
 - **Start Date:** Sep 2017 - current
-
- Undergraduate Courses Taught:
 - CSE 1341 – Principles of Computer Science - Java
 - CSE 3342 – Programming Languages – C++, Python, Java, CLisp, Prolog, Node.js, Angular, etc.
 - Graduate Courses Taught:
 - o CSE 7316 – Requirements Engineering
 - o CSE 7319 – Software Architecture and Design
 - o CSE 8313 – Object-Oriented Analysis and Design

Introduce yourself:

- Your name
- Where are you from?
- Your programming experience..
- Anything you want to share about yourself – fun facts (optional)

Class Information

- Daily Poll or Survey
 - A brief survey to gauge your thoughts on the class's progress.
- All classes Recorded
 - Sessions are recorded for later viewing.
- 10 minute break every 45-60 minutes.

Program Overview

WEEKS 1-4

programming fundamentals



- Learn object-oriented programming to compose larger programs together in Java
- Work with development tools like Eclipse, and Git

WEEKS 5-8

databases and APIs



- Store and retrieve data using the Postgres relational databases
- Consume and share data from our applications over the Internet using APIs

WEEKS 9-12

front-end programming



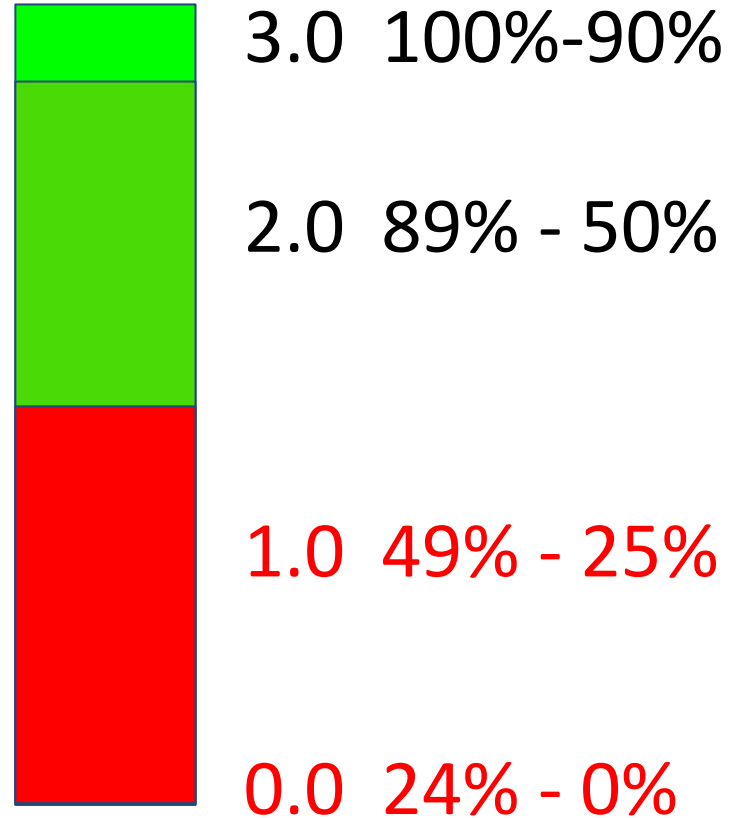
- Develop web application interfaces using HTML, CSS, and JavaScript
- Learn how create web components using the Vue.JS JavaScript framework

Daily Workflow



1. Before each lecture
 - a. Complete the Reading **AND Tutorial** in Bootcamp OS/LMS
 - b. Complete the Quiz
2. Lecture
 - a. Starts at **9am EST**. Be on time.
 - b. Ends when we complete that day's objectives
3. After Lecture
 - a. Exercises
 - b. Pathway (check Google Calendar)

“Grading”



Academic Standing

- You need to maintain an exercise score average of 2.0 or higher to remain in good academic standing.
- You can always check your average by logging into Bootcamp OS / LMS.

^ Intro to Programming in Java ●●● Average: 2.7

Exercise Title	Score	Due Date
Variables and data ty... Learn about declaring and initializing variables in Java.	3/3	
Logical branching Learn about boolean expressions and running code conditionally.	2/3	Due: 03/08/2022 11:00 EST
Loops and arrays Learn about loops and arrays in Java.	3/3	Due: 03/09/2022 5:00 EST







Week 1 Overview

Monday

Intro to Tools

Tuesday

Variables and
Datatypes

Wednesday

Logical
Branching

Thursday

Loops and
Arrays

Friday

Command
Line Programs

Objectives

1. The File System
2. Windows Terminal – (Bash Shell)
3. GIT

File System

The way a computer organizes data. Includes Folders (or Directories) and Files

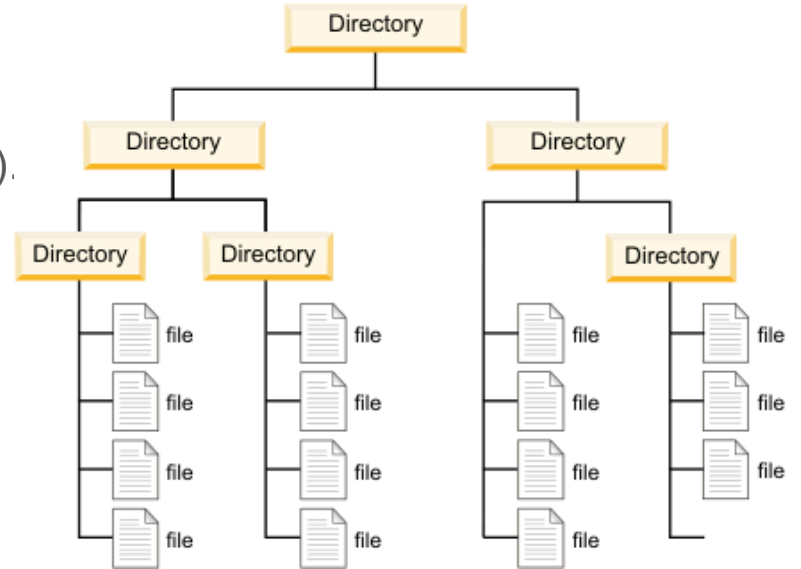
Organized into a tree structure. All folders are in a another folder, except the ROOT folder (starting folder). All files must be in a folder.

All folders can contain 0 to many folders and/or files.

folder (directory) - purely for organization

files - content (pictures, documents, etc.)

Folders and Files have *metadata* (properties) associated with them like date modified, names, and permissions (who can do what with that folder or file)



Windows Terminal – (BASH Shell)

What is Windows Terminal?

In Windows, a terminal refers to a command-line interface (CLI) application that allows you to interact with the operating system through text commands. It provides a text-based environment where you can execute various commands, run programs, and perform tasks without relying on a graphical user interface (GUI).

Windows operating systems include a default terminal application called Command Prompt (cmd.exe) for a long time. However, starting with Windows 10, Microsoft introduced a more powerful and feature-rich terminal application called Windows Terminal. Windows Terminal supports multiple command-line environments, including Command Prompt, PowerShell, and Windows Subsystem for Linux (WSL).

Using a terminal in Windows, you can perform various tasks such as navigating the file system, managing files and directories, executing programs, running scripts, configuring system settings, and much more. It provides a flexible and efficient way to interact with the operating system, particularly for system administrators, developers, and power users who prefer working with commands rather than a graphical interface.

BASH Shell

What is a Shell?

A shell provides a text-based interface to interact with a computer's file system. In Shell you write lines of code and commands that the computer understands to direct the computer to do what you want it to do.

What is BASH (Bourne-again SHell)?

There are many types of shells available on computer systems. Each Shell has its own set of commands. BASH is a commonly used shell that is available on Mac OS, Linux, and Windows. While Windows 10 does have a built in BASH Shell, GIT BASH provides a more robust version of the shell.

Why use a shell?

- Many programming tasks are more flexible and faster in the Shell
- It can be scripted, which means we can write our own reusable commands
- Many programming tools do not have graphical user interfaces (GUI), or have more options available in the shell version. Often the GUI for these tools and just for convenience and issue shell commands in the background, so if we don't understand the shell commands, then we can't troubleshoot problems we may encounter.

Shell/Terminal Basics

The **Root** directory is where the file system begins for the computer. All directories and files on the computer are in the root directory (/).

The **home directory** is where the file system that has been assigned to your user begins. The home directory can always be accessed by the `~` alias.

The **working directory** is the directory that is currently being accessed.

Shells are controlled using **commands** that tell the computer what you want it to do.

Many shell commands take **arguments**, which are extra parts of the command that change its behavior.

Basic Shell Commands

`.` is an alias for the current directory

`..` is an alias for the parent directory of the current directory.

Command	Description
<code>cd</code>	change directory. <code>~</code> specifies the home directory. (<code>cd ~</code>)
<code>pwd</code>	Tells you where you are in the file system
<code>ls</code> <code>ls -la</code>	list all files and folders in a directory. <code>-a</code> (shows all files and folders) <code>ls -a -l</code> (shows details about the files and folders) <code>ls -l</code> . can be used together as <code>ls -la</code> In file details first letter is <code>d</code> for a directory and <code>-</code> for a file.
<code>mkdir <dir name></code>	Creates a new directory
<code>code <filename></code>	Creates a file then opens it in Visual Studio Code, or if the file exists it opens it.
<code>touch <filename></code>	Creates an empty file
<code>cp <file> <new location></code>	Copies a file
<code>mv <file> <new location></code>	Moves or renames a file
<code>cat</code>	print contents of a file on the screen
<code>rmdir <dir-name></code> <code>rm -r <dir-name></code>	Removes directory only if it is empty Removes folder with <code>-r</code> (recursively remove all contents). <code>-f</code> (force) - make it happen no matter what (no prompt) - dangerous .

Command Line Commands: Moving Around

- Data in your workstation are organized into files and folders.
- The main command to move around folder is **cd**. There are several variations of these:
 - **cd ~** : Returns you to your home directory.
 - **cd <directory name>** : Takes you to a specified directory i.e. cd workspace takes you to a folder called workspace
 - **cd ..** : Takes you one level up.
 - **cd /** : Takes you to the root
- You can always see what directory you're in by typing **pwd** (print working directory).
- The **ls** command lists all the files in the current directory.
- The **ls -al** command will list all the files, including any hidden ones.

Moving Around: Absolute Path

- When you used the `pwd` command, the output would have looked something like this:

```
Student@DELL-JAVA MINGW64 ~/workspace  
$ pwd  
/c/Users/Student/workspace
```

Recall that `pwd` displays the current directory. Note that the response from this command is an absolute path since it starts with a slash (/).

Moving Around: Relative Path

- A relative path is differentiated from the absolute path by the absence of the initial slash:
 - **cd /c/Users/Student/workspace** uses an absolute path to get me to the workspace folder.
 - Alternatively, if I were already in my respective user folder (Student), typing **cd workspace** uses a relative path to get me to the workspace folder.

Moving Around: The Tilde (~)

- The tilde (~) is a special symbol or alias used to denote the home directory. For all of your workstations this has been set to: `/c/users/Student`

```
Student@DELL-JAVA MINGW64 ~/workspace
```

```
$ cd ~/workspace
```

Therefore, the above command will take you to: `/c/Users/Student/workspace/`

Making Directories

- To create a directory we use the **mkdir <filename>** command.
- **\$ mkdir testDir**

Command Line Commands: Copying

- To copy a file from 1 directory to another: `cp <source> <destination>`

```
Student@DELL-JAVA MINGW64 ~
```

```
$ cp ~/testdir/file.txt ~/othertestdir
```

- To move a file from 1 directory to another: `mv <source> <destination>`

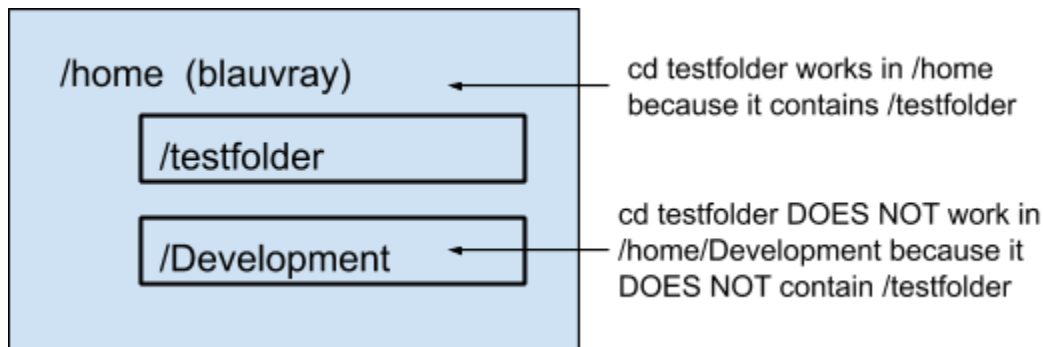
```
Student@DELL-JAVA MINGW64 ~
```

```
$ mv ~/othertestdir/file.txt ~/testdir/
```

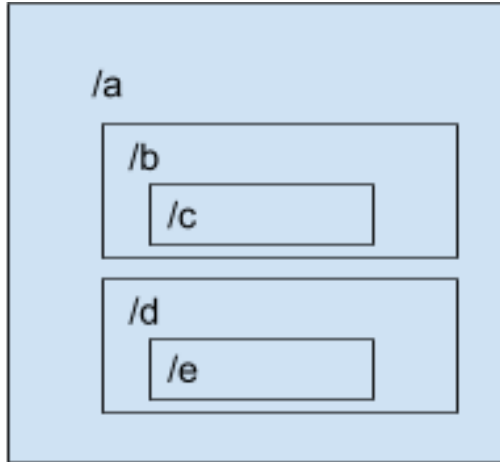
- Copy and Move differ in that the latter will remove the file from the source. With copy, the source retains a copy of the file.

Paths - Review

- A **Path** is text representation of where a directory or file is located in the File System. Paths can be **absolute** or **relative**.
 - An **absolute path** is one that starts at the **Root** of the file system. It points to the same location in the file system regardless of where it is used. (Starts with /)
 - A **relative path** is one that starts *relative* to the current working directory (location). It points to a different location in the file system depending on where it is used. (Starts with a folder or file name.)



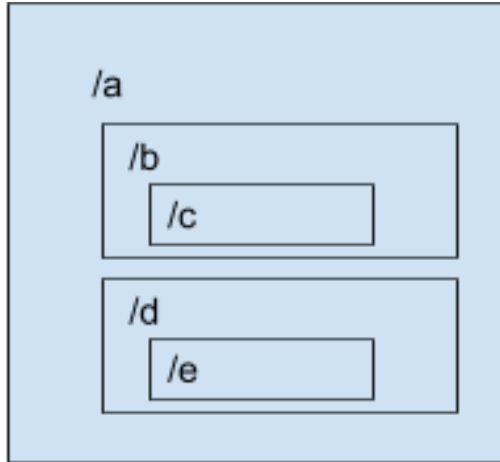
If we had the following file system, and our current location was `/a/b/c` in `~testFolder`



To change directory with the `cd` command:

1. What relative path would change to `/a/b`?
2. What relative path would change to `/a`?
3. What absolute path would change to `/a`?
4. What relative path would change to `/a/d/e`?
5. What absolute path would change to `/a/d/e`?

If we had the following file system, and our current location was `/a/b/c` in `~testFolder`



To change directory with the `cd` command:

1. What relative path would change to `/a/b`?
`cd ..`
2. What relative path would change to `/a`?
`cd ../..`
3. What absolute path would change to `/a`?
`cd /c/Users/Student/testFolder/a`
`cd ~/testFolder/a`
4. What relative path would change to `/a/d/e`?
`cd .././d/e`
5. What absolute path would change to `/a/d/e`?
`cd /c/Users/Student/testFolder/a/d/e`
`cd ~/testFolder/a/d/e`

Terminal / Shell Tips

1. **No news is good news.** Many commands only give feedback when there is a problem.
2. **A history of commands is kept.** You scroll through past commands using the Up and Down arrow keys.
3. **The tab key can be used to autocomplete a path.** It only completes if you have enough unique characters for it identify the directory or file you are trying to identify.
 - a. You can press tab every few letters to autocomplete the path to save time and avoid spelling errors.



git



What is git?

git is a Version Control software.

Version Control is software that records changes to a set of files over time, so previous version can be recalled.

Used for coordinating work among programmers collaboratively developing source code during software development.



More on Version Control

- Version control software allows developers to save and version their code.
- Version control is Source Control – they can be used interchangeably.
- In this class, we will be using git/GitLab.
- Git is an example of a distributed source control system, where a repository exists locally on your own workstation and on a central network location.
- To set up a git repo (repository) on a machine, you clone the repo from GitLab. Cloning pulls all the commits from the remote repo to your local machine.

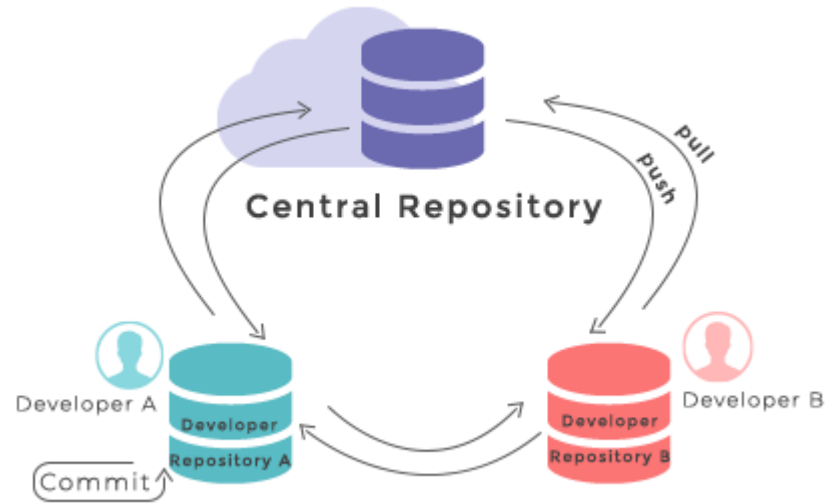
Why use git?

1. Teams can go back in time before a bug occurred to compare to a working version
2. Sharing code among team members is simpler
3. Team members can develop code in parallel, and then merge their code together when complete
4. Once you have *committed* your work to GIT. It is very difficult to lose it, even if you make a mistake!
5. Let developers try things without the risk of losing previous work.

git Repositories

A git Repository is a collection of files and history that are versions of a project.

A remote central repository is used to store the project and each developer working on it has a local repository on their computer that is a copy of the remote repository. Changes are then shared between each developer and the remote repository allowing multiple developers to collaborate on a project.



Remote Repositories

upstream

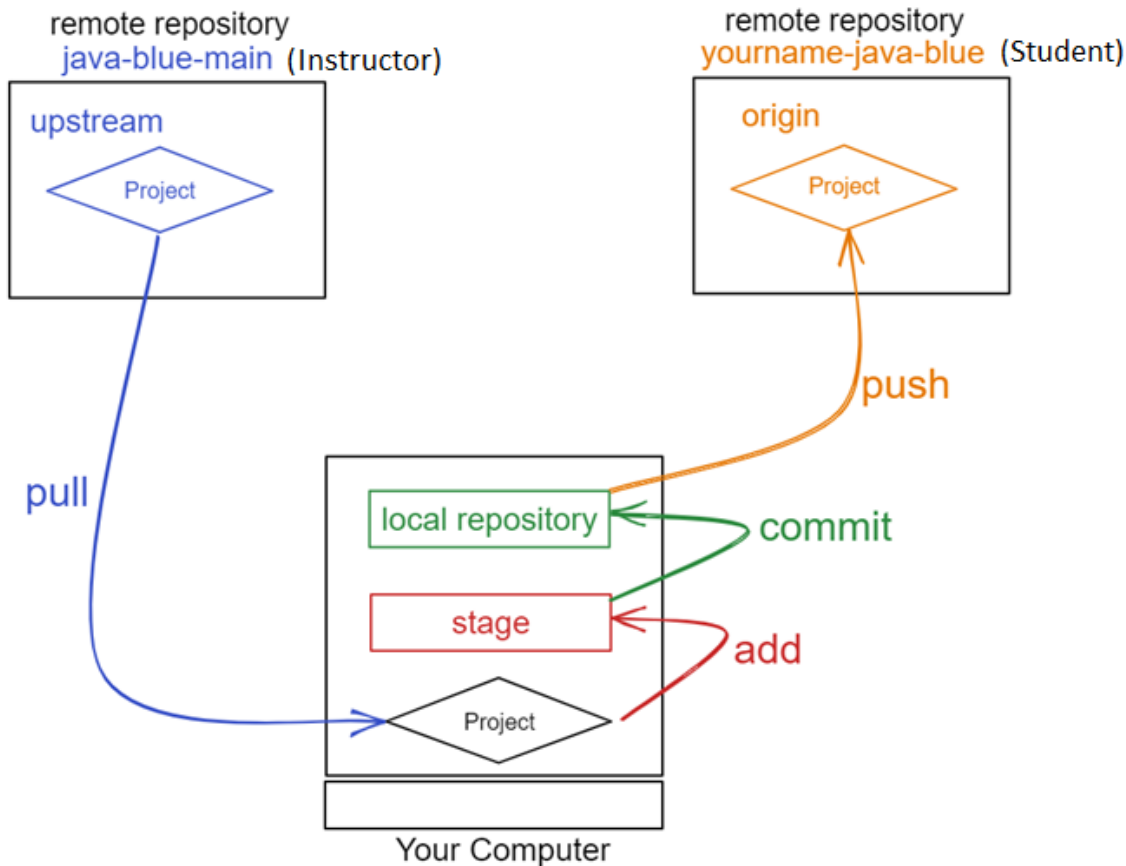
java-purple-main

Instructor repository used to get new materials and exercises. Can only pull.

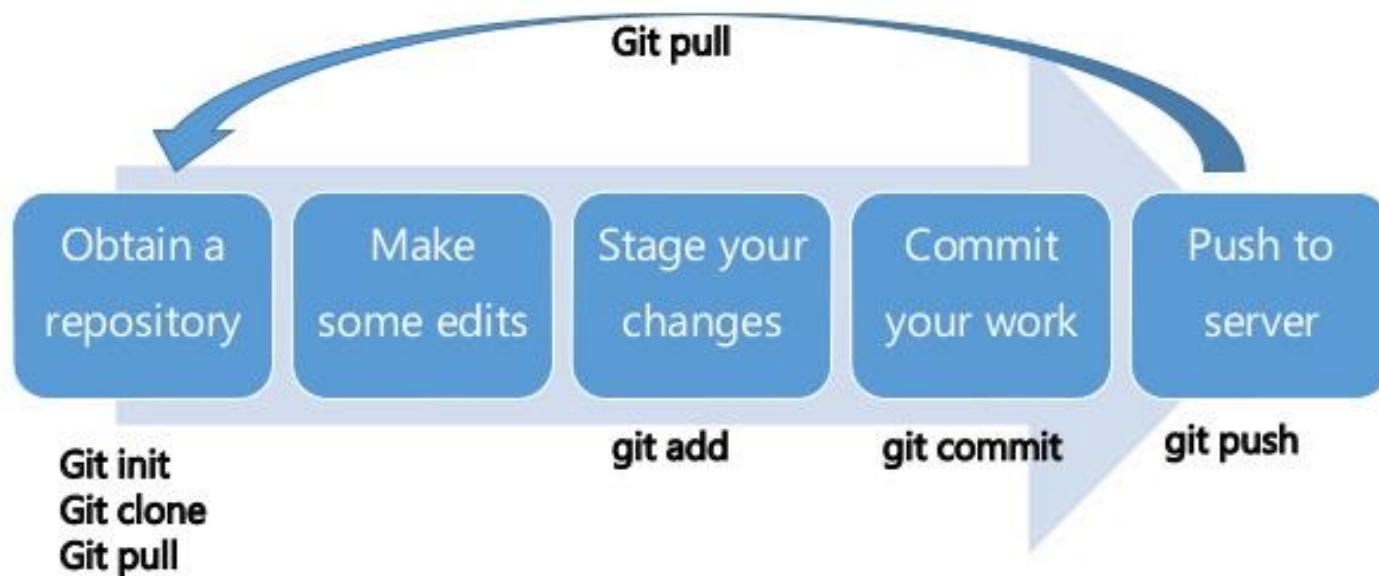
origin

yourname-java-purple

Student repository used to backup and submit materials and exercises. Can push and pull.



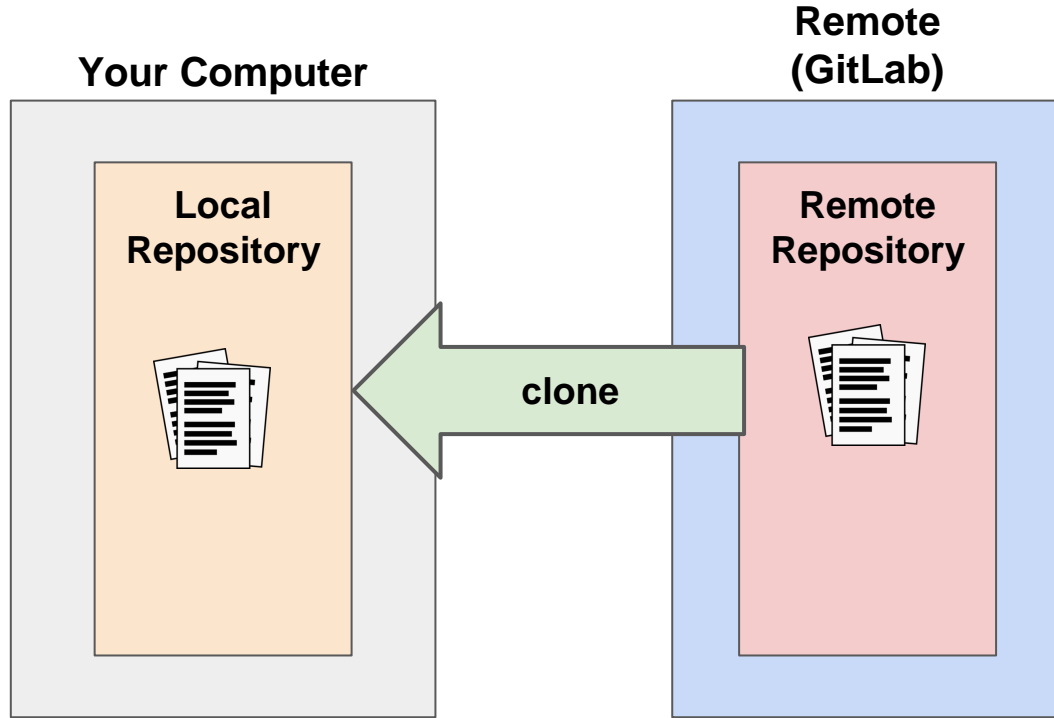
Git usages : Understanding Git Workflow





Commands

git clone



clone makes a copy of a remote repository as a local repository on your computer as a starting point.


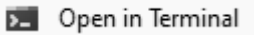
This copy includes all the files and history from the remote repository.



Cloning your student repository



Open Your Working directory

1. Open Windows File Explorer 
2. Under *This PC* select *C:*
3. Then User → Student → workspace (`C:\Users\Student\workspace`)
4. Right click your student repo and select 'Open in Terminal'.
5. This will open the Windows terminal in this folder.  Verify that it shows your `student repo` at the end of the first line.

Getting to your Repository

In the Chrome Browser go to

`https://git.techelevator.com/`

Login using your GitLab account

Click on the Repository named:

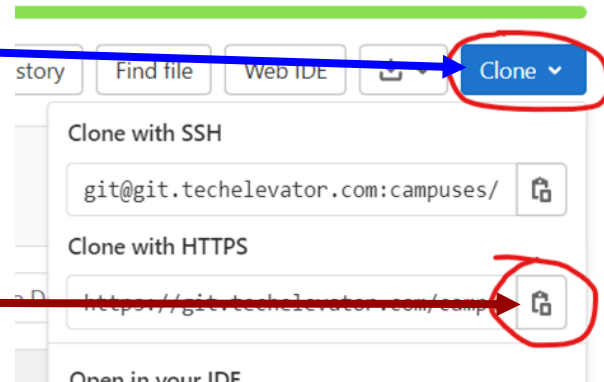
Your Name Student Code

Where Your Name is your full first and last name.



Cloning your Repository

1. On the repository page click the **Clone** button on the upper right.
2. Click the **Copy to Clipboard** button beside Clone with HTTPS



1. Return to the Windows Terminal and type `git clone`, then right-click and select paste

```
Brian@DESKTOP-CMC2D8C MINGW64 ~/source/repos
$ git clone https://git.techelevator.com/campuses/cbus/jan-2022/java-blue/student-code/abdinasir-atto-student-code.git
```

1. Once the address has been pasted press the Enter key. **You will need to enter your GitLab username and password.**



Change to your local repository

1. Once the clone completes, type `cd your-name-student-code`, where *your-name* is your first-last name, and press Enter

```
Brian@DESKTOP-CMC2D8C MINGW64 ~/source/repos  
$ cd abdinahir-atto-student-code/
```

1. Verify you are now in your student-code folder

```
Brian@DESKTOP-CMC2D8C MINGW64 ~/source/repos/abdinahir-atto-student-code (main)  
$
```



Run the setup Script

Run the setup script by typing **sh setup.sh** and pressing enter, it will ask for your first and last name and email address.

Use your real name and email. Potential employers will see this information!

sh setup.sh

```
Brian@DESKTOP-CMC2D8C MINGW64 ~/source/repos/abdinasir-atto-student-code (main)
$ sh setup.sh

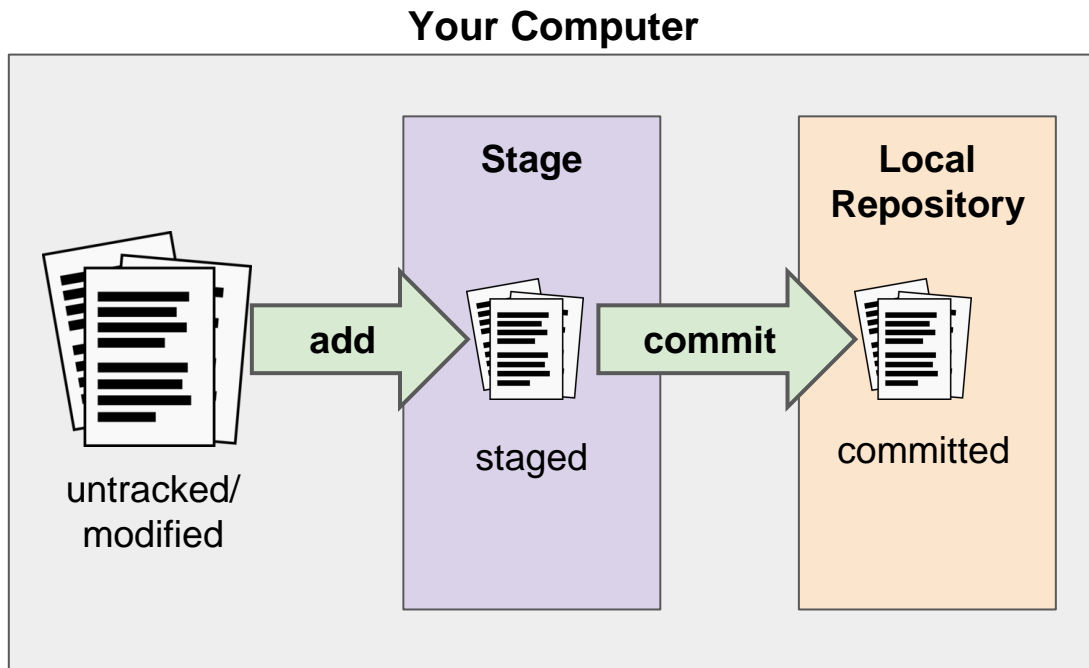
Enter your name (First Last): Brian Lauvray
Enter your email: brian@techelevator.com

Setting Up Global Configuration Settings
Setting up Git Editors and Tools...

Configuring Upstream...
Done.
```

git status

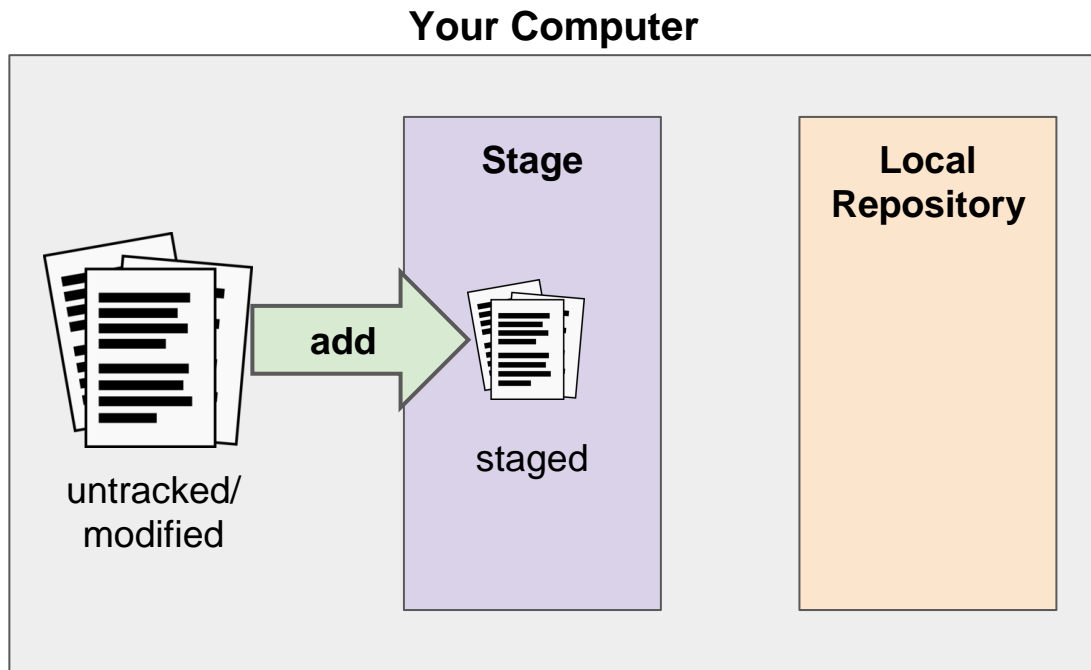
status reports the state of the files in the directory and what stage in the process each is in.



File Status

untracked	File/Folder can be seen by GIT but is not being tracked by GIT
modified	A file/folder has been changed, but not staged
staged	A change has been added
committed	A change has been added to the local repository

git add -A



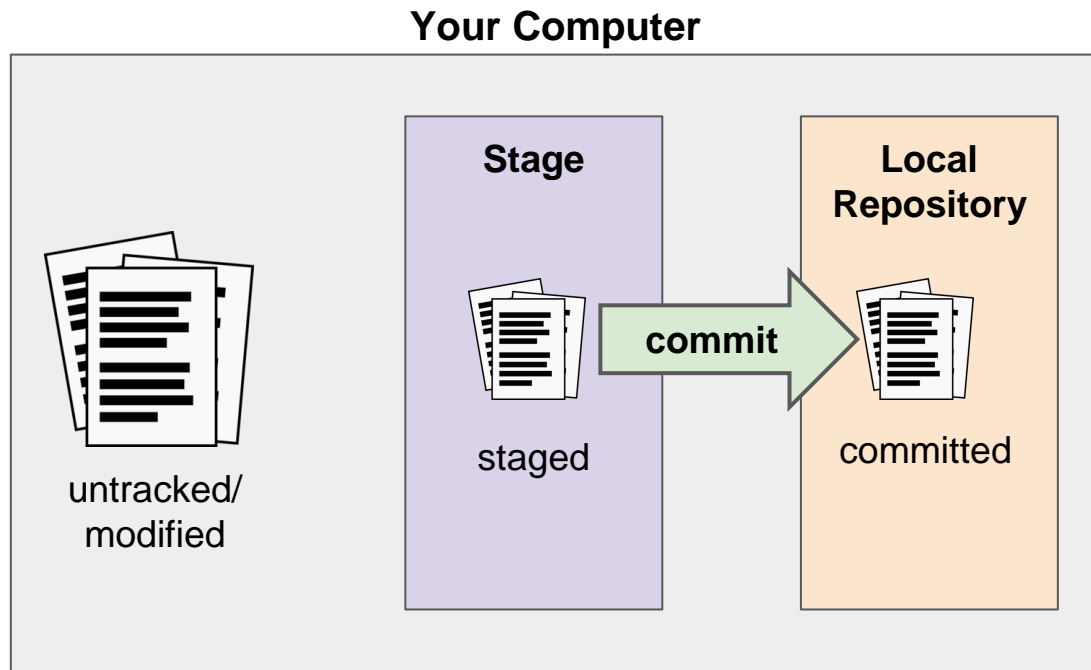
add adds a set of files to the stage.

The *stage* is an area used to prepare for a commit, which allows more control over smaller commits.

The files to add must be included in the command. - A tells git to add all changes, including deletes.

git add -A

git commit



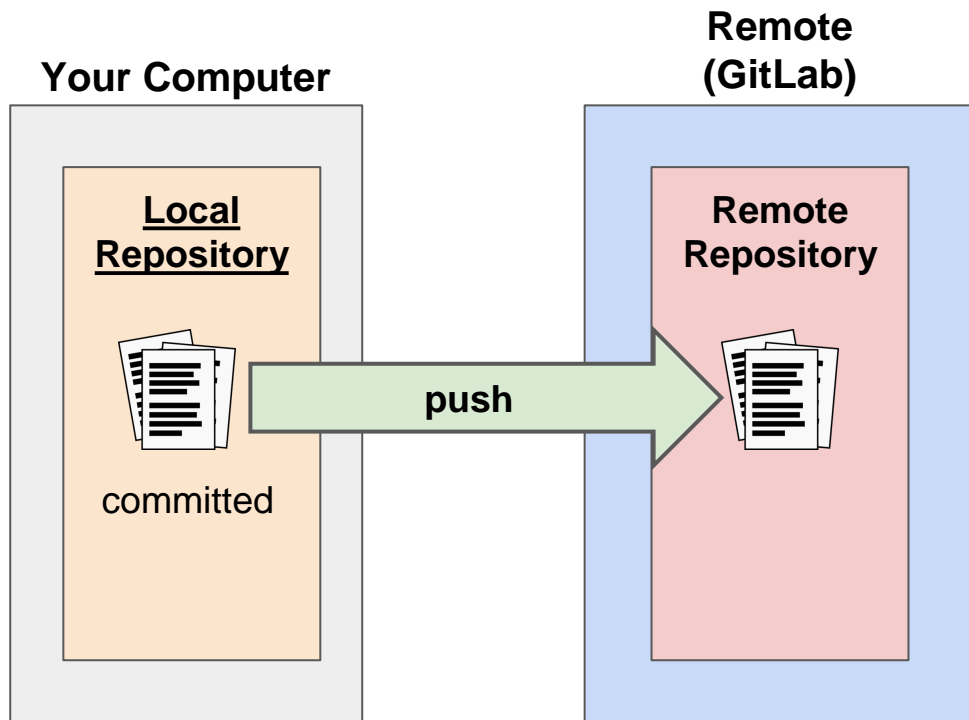
commit moves a set of changes from the stage to the local repository.

Committing is like saving the changes. When committed a history is created and changes become very difficult to lose.

Commit requires a message that describes the changes.

git commit -m “commit message”

git push



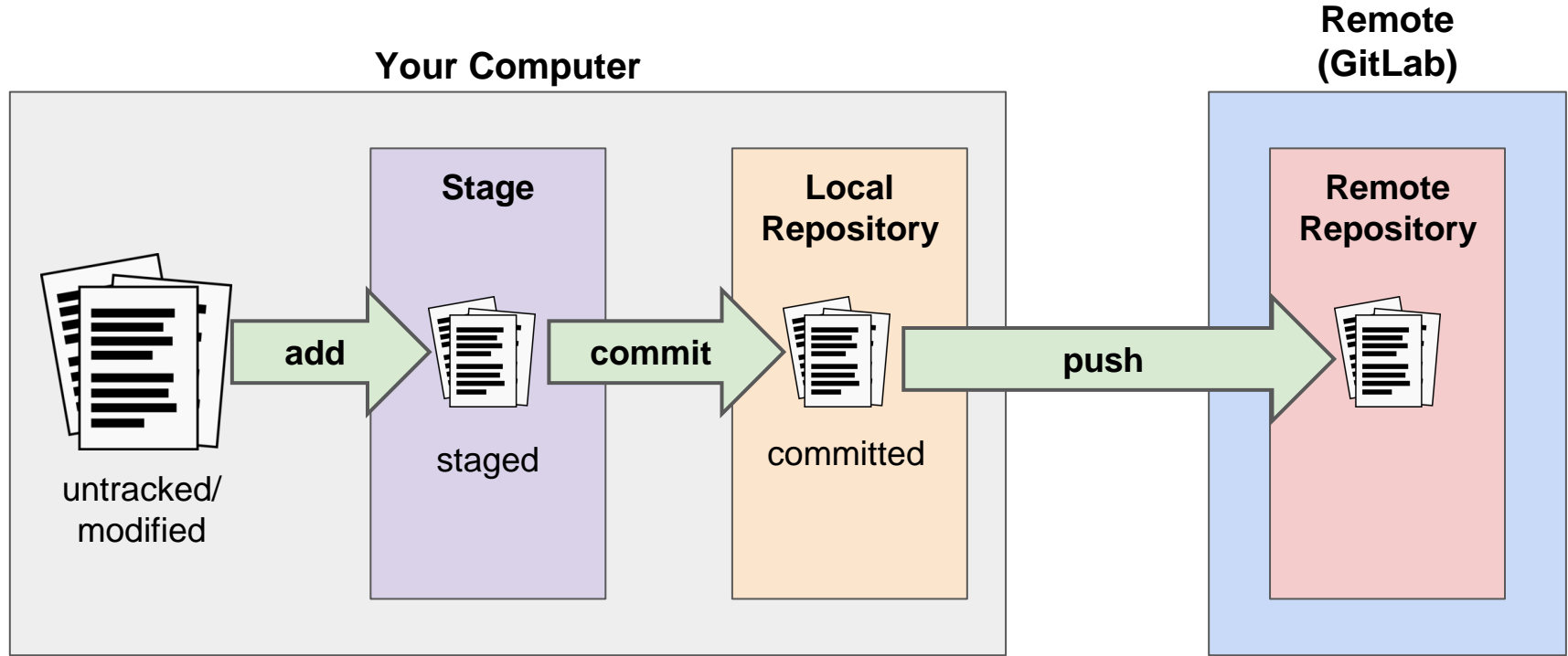
push copies the changes and history from the local repository to the remote repository.

Once in the remote repository it is available for other developers on the team to access.

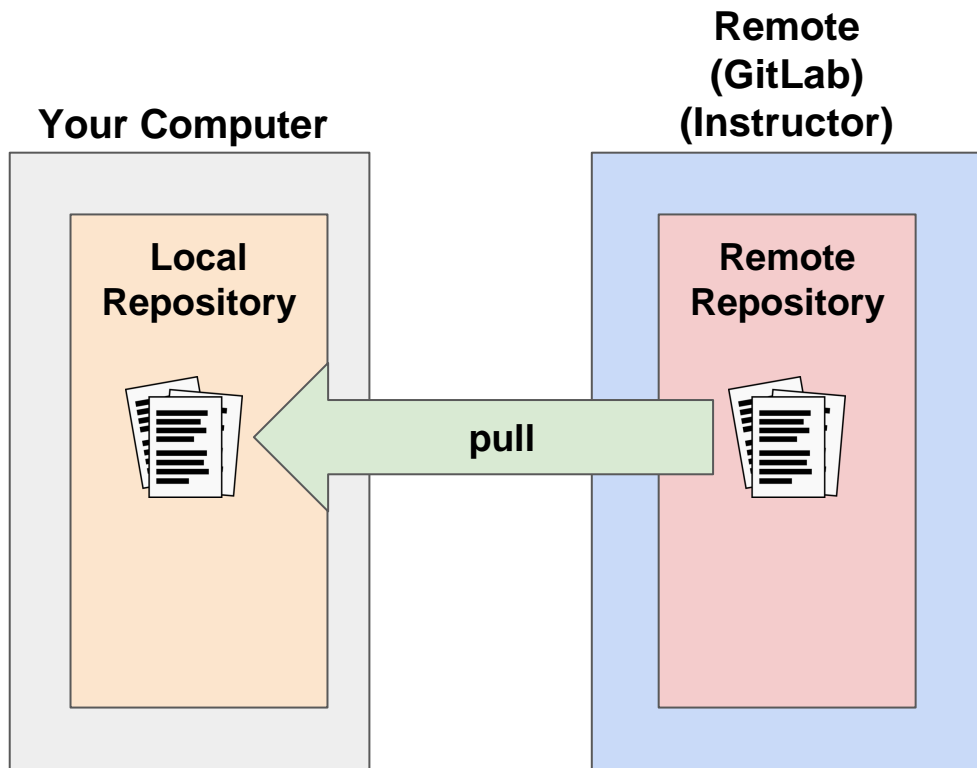
push requires name of the **remote repository** and the **branch**.

git push origin main

Workflow when using git



git pull



pull copies the changes and history made by other developers from the remote repository into the local repository.

pull requires name of the **remote repository** and the **branch**.

git pull upstream main

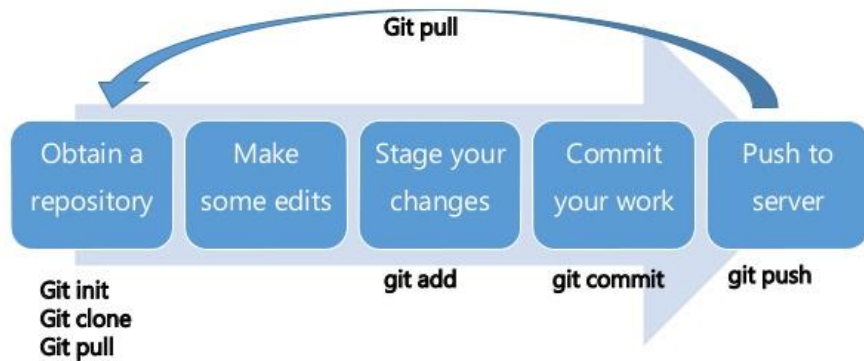


Cheatsheet

Commands

clone	Get a copy of a repository as a starting point
add	Add changes to the stage
commit	Add changes to the local repository
push	Copy changes from the local repository to the remote repository
pull	Copy changes from the remote repository to the local repository
status	tells us the state files are in in this process

Git usages : Understanding Git Workflow



File Status

untracked	File/Folder can be seen by GIT but is not be tracked by GIT
modified	A file/folder has been changed, but not staged
staged	A change has been added
committed	A change has been added to the local repository



Mantra

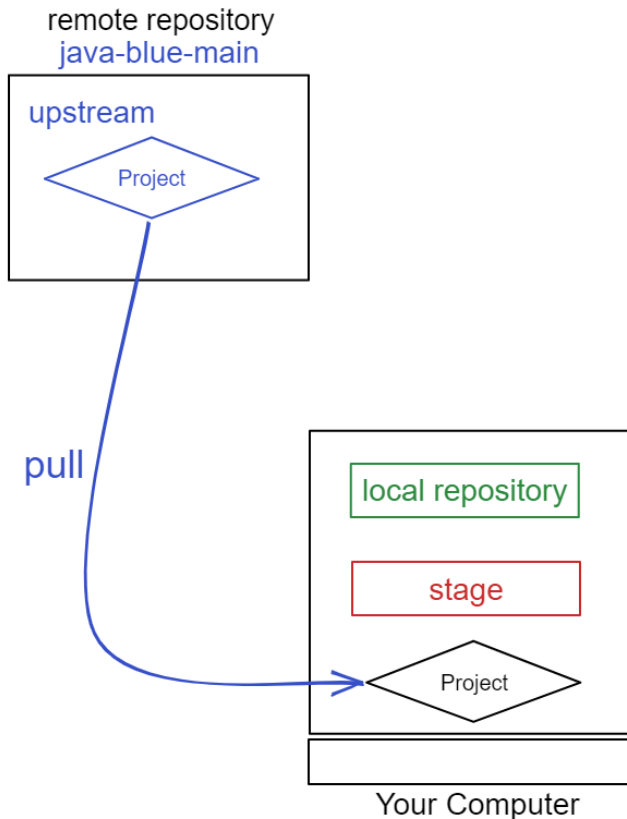
Commit Early, Commit Often



git

Daily Workflow

get the new materials from upstream (instructor's repo)



```
$ cd ~/source/repos/<yourname>-java-yellow  
$ git pull upstream main
```

Watch the output of the command.

If it says it was not able to pull the changes, then you may need to add and commit first.

1. `$ git add -A`
2. `$ git commit -m "your comment"`
3. `$ git push origin main`



Pull from upstream

Type ***git pull upstream main*** and press enter to pull recent changes from the upstream repository that was just configured.

git pull upstream main

```
Brian@DESKTOP-CMC2D8C MINGW64 ~/source/repos/abdinasir-atto-student-code (main)
$ git pull upstream main
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 282 bytes | 21.00 KiB/s, done.
From https://git.techelevator.com/campuses/cbus/jan-2022/java-blue/instructor-code
 * branch            main              -> FETCH_HEAD
 * [new branch]      main              -> upstream/main
Updating e5a63f7..3842137
Fast-forward
 welcome-to-java-blue.txt | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 welcome-to-java-blue.txt
```



Verify it worked

Type **ls** and press enter

ls

you should see module-1 listed.

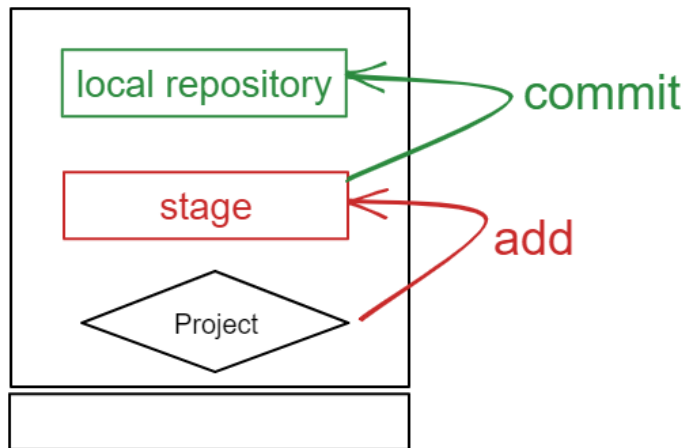
```
Brian@DESKTOP-CMC2D8C MINGW64 /d/TechElevator/Cohort17/brian-lauvray-student-code (main)
$ ls
README.md module-1/ setup.sh*
```



If you see the module-1 directory listed then you are done and your configuration was successful!

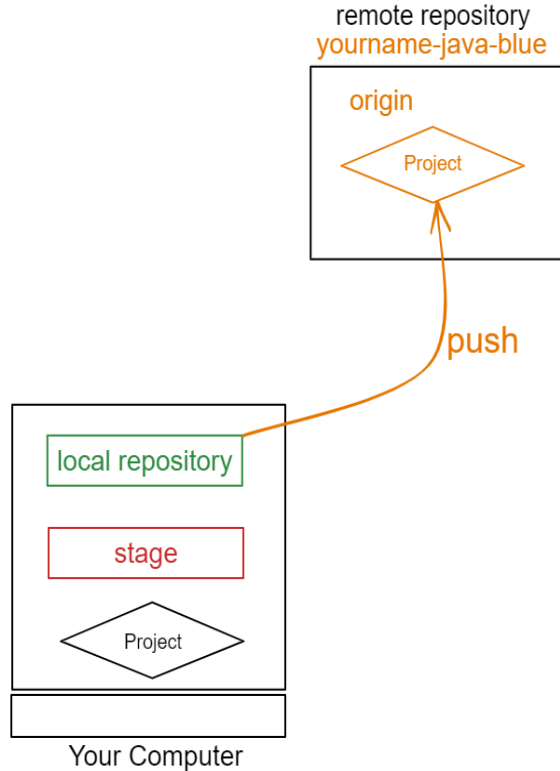
complete the exercises

1. `$ cd ~/source/repos/<yourname>-java-purple`
2. `$ git add -A`
3. `$ git commit -m "commit message"`



**As you work *add* and *commit*
OFTEN!**

submit your work to origin



1. `$ cd ~/source/repos/<yourname>-java-yellow`
2. `$ git add -A`
3. `$ git commit -m "commit message"`
4. `$ git push origin main`

Watch the output for failures! If you see a failure, reach out to an instructor ASAP.

You can verify what has been submitted by looking at the file in GitLab.

You can push as often as you like. Only the last changes made will be reviewed.

Remember the Workflow!

1. Get new materials (usually only once per day)

```
git pull upstream main
```

2. complete the exercises

3. Add your changes to the stage

```
git add -A
```

4. Commit the changes to your local repository

```
git commit -m "comment"
```

5. Repeat 3 & 4 *often* as you work on your exercises!

6. Push your changes to GitLab to backup and/or submit. (as often as you like)

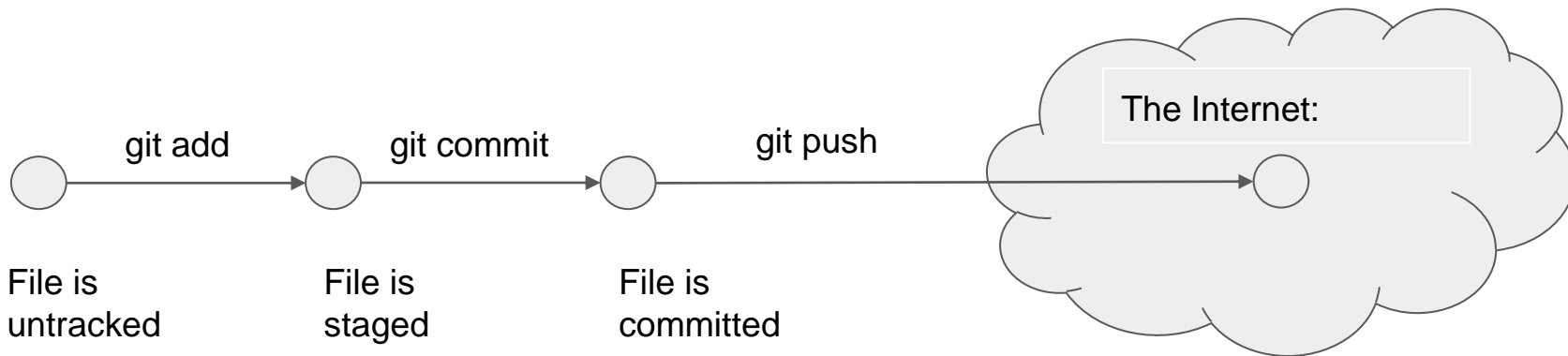
```
git push origin main
```

These commands should **ALWAYS** be run in your local repositories root:

```
~/source/repos/yourname-java-purple
```

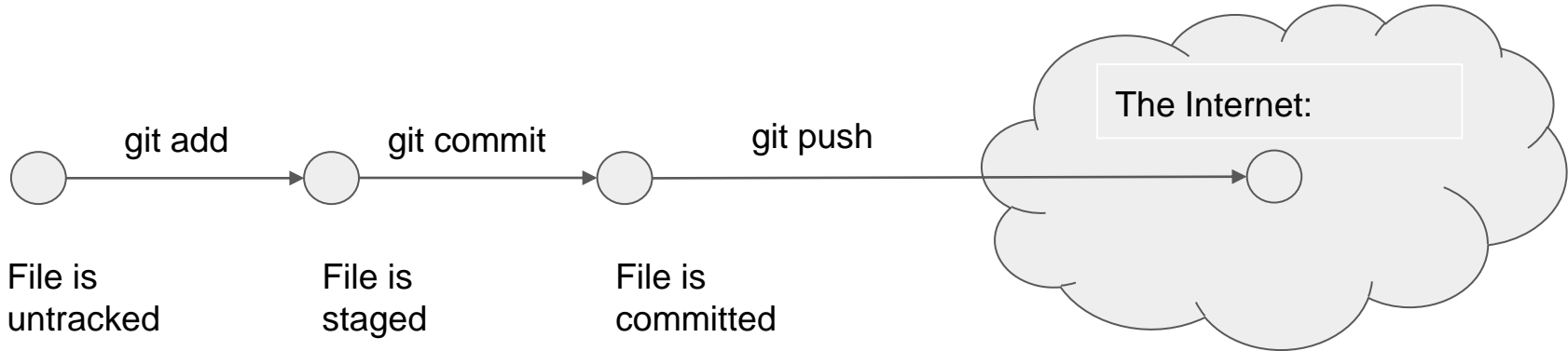
Source Control : Git Flow () - Example

- **git status**: See the current status of your files.
- **git add -A**: Stage any files you have changed.
- **git commit -m "Commit message"**: Commit files to your local repository
- **git push origin main**: Push committed changes to network repository.



Source Control : (Example)

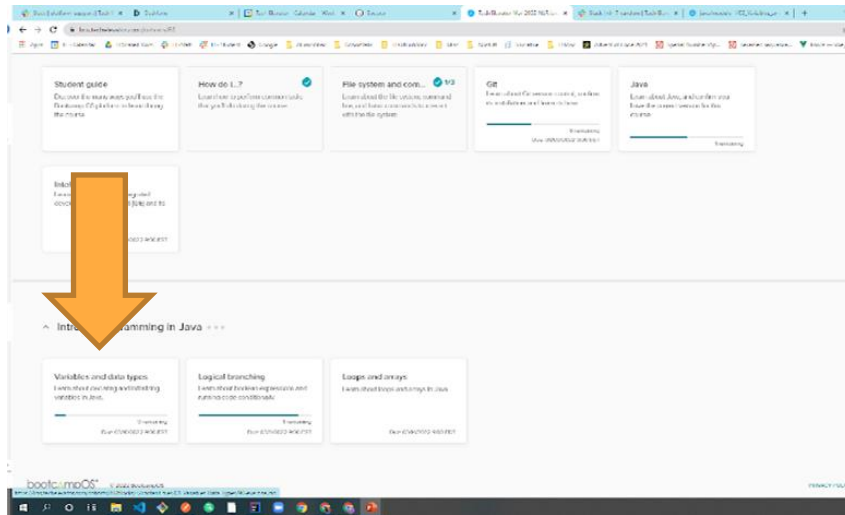
- **git add about-me.txt**: Stage the about-me.txt file you have changed.
- **git commit -m "Add changes to about-me"**: Commit file to local repo
- **git push origin main**: Push committed changes to network repository.



Things to keep in mind:

- You want to pull often:
 - Pull when your instructors ask you to.
 - Pull first thing in the morning when you get to class.
 - Pull before you plan to push an assignment.
- Grading takes place by you pushing to your student-code folder in GitLab and then submit it in Bootcamp OS (BOS – lms.techelevator.com).

How to submit your exercises



- Open a new tab in Chrome and navigate to lms.techelevator.com
- Find the assignment

How to submit your exercises

The screenshot shows the Tech Elevator web application. At the top, there's a navigation bar with the Tech Elevator logo and a sidebar menu on the left. The main content area is titled "Exercise" and displays a "1" in a circle, indicating the first of three exercises. Below this, there's a "Continue Assessment" button. A large orange arrow points to the "RUN TESTS" button at the bottom of the exercise card. The exercise card also includes a list of instructions and a text input field containing the URL "https://git.techelevator.com/campuses/nir/mar-22/java-green/student-code".

Exercise

1

Continue Assessment

1. Find the `README.md` file in the `exercise` folder for this unit.

2. Open the `README.md` file in your preferred Markdown viewer.

3. Follow the instructions in the `README.md` file to complete the exercise.

4. When all tests pass, commit and push your changes.

5. In your browser, navigate to the current exercise in your repository.

6. Copy and paste the URL from the browser into the textbox below.

7. Click **Submit** to submit your exercise.

<https://git.techelevator.com/campuses/nir/mar-22/java-green/student-code>

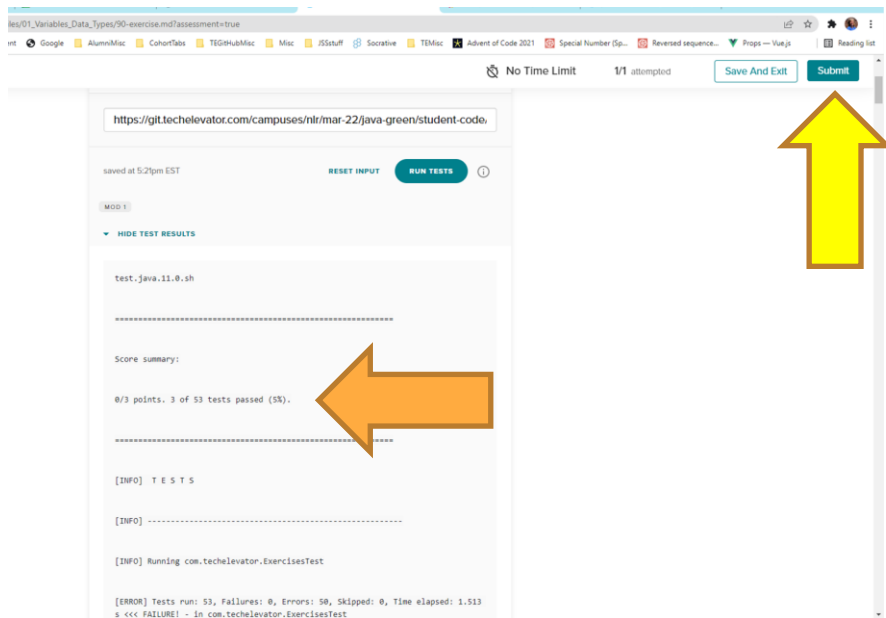
RESET INPUT RUN TESTS

SHOW TEST RESULTS

That's the end! When you're ready, submit this assessment at the top of the page.

- Click “Run Tests”

How to submit your exercises



- Check the screen for your score summary
- Don't forget to hit the submit button!

