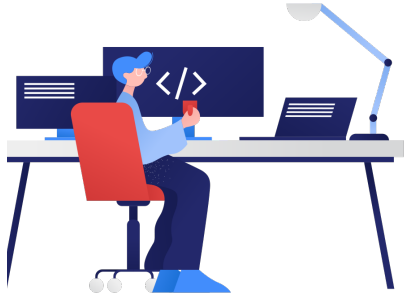
CSE 15L: SOFTWARE TOOLS AND TECHNIQUES

Instructor: Onat Gungor

SCHEDULE

Last Week: Guest Lectures



Today: Course Review

REMOTE ACCESS

$ ssh cs15lsp22zz@ieng6.ucsd.edu

Replace **zz** with the **letters** in your course specific account

Format: $ ssh user@hostname

Press 'Enter'

First time connection message

The authenticity of host 'ieng6.ucsd.edu (128.54.70.227)' can't be established. RSA key fingerprint is SHA256:ksruYwhnYH+sySHnHAtLUHngrPEyZTDI/1x99wUQcec. **Are you sure you want to continue connecting (yes/no/[fingerprint])?**

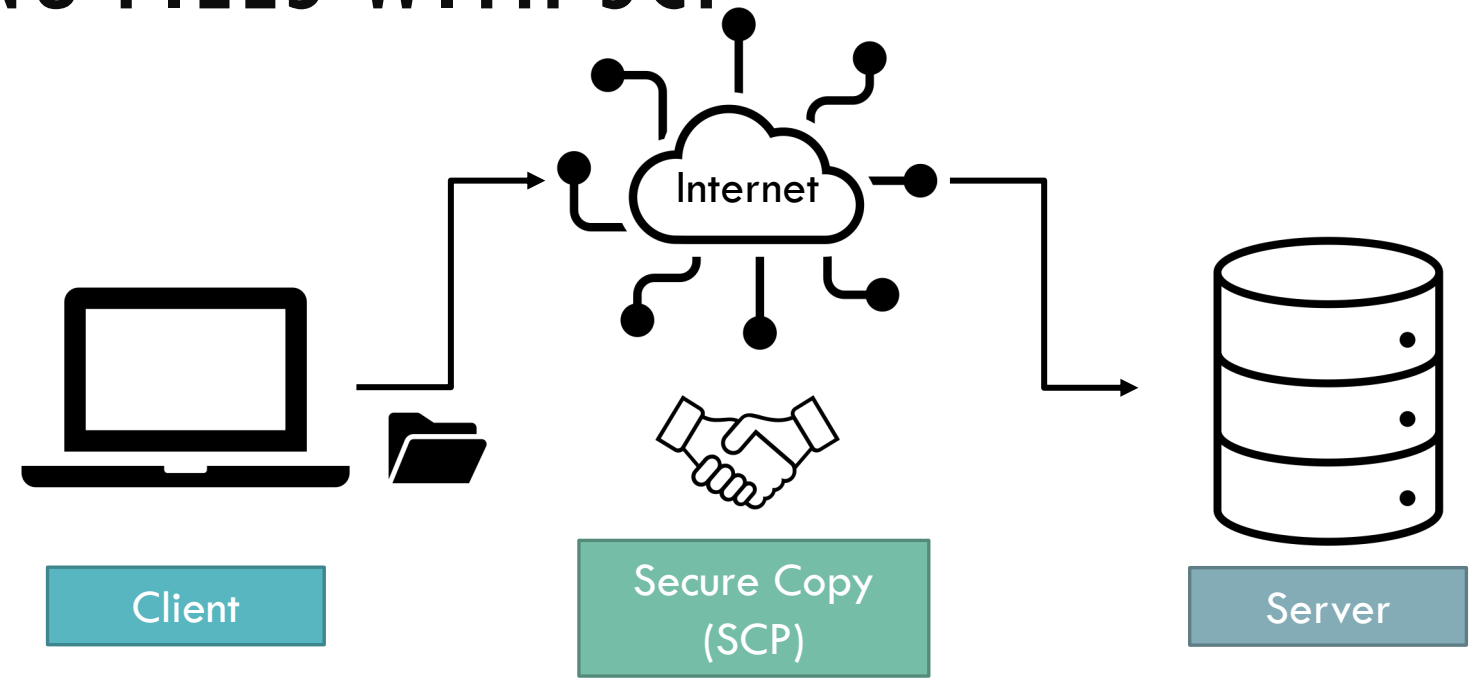
Type 'yes' and Press 'Enter'

$ (cs15lsp22zz@ieng6.ucsd.edu) Password:

Enter your password

Hello cs15lsp22zz, you are currently logged into ieng6-203.ucsd.edu

MOVING FILES WITH SCP



$ scp [*user1*@*hostname1*:]*file1* [*user2*@*hostname2*:]*file2*

user1, user2: login account to use on the remote host
hostname1, hostname2: names of the remote host from or to which the file is to be copied
file1: file name or directory name to be copied
file2: destination file name or directory

$ scp HelloWorld.java cs15lsp22zz@ieng6.ucsd.edu:~/

Enter your password

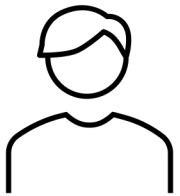
Once Onat does this, he can ssh or scp without entering his password.

SSH KEYS

Every time I use ssh or scp, I must type my password.

This is super frustrating and time consuming. Why don't you use **ssh keygen**?

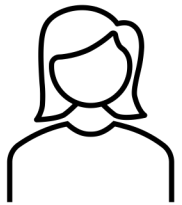
ONAT



That sounds great! Can you show me how I can set up these keys?

ssh keygen creates public and private keys which are stored on the server and client.

MARY



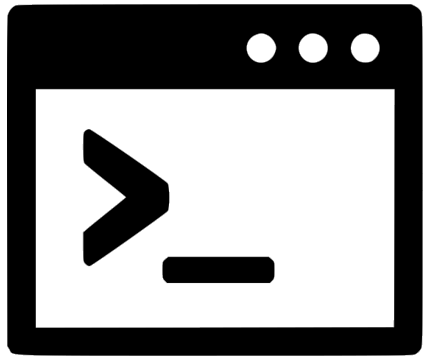
Public key is copied to server's **~/.ssh/authorized_keys** directory.

Then, we need to copy the public key to the .ssh directory of the server.

Type command **ssh-keygen** which will then ask you: 1)directory to save, 2)passphrase.

Keys are generally stored in client's **~/.ssh** directory.
private key: id_rsa
public key: id_rsa.pub

COMMON UNIX COMMANDS



- **`pwd:`** print working directory
- **`ls:`** list files
  - ls [options] [names], e.g., ls -l, ls –a, ls –lat
- **`cp:`** copy
- **`mv:`** move or rename
- **`cd:`** change directory
- **`mkdir:`** make directory
- **`rm:`** remove
- **`cat:`** view or create a file
- **`touch:`** create a file
- **`man:`** display command manual

- Some commands can be run as is: **`pwd, ls`**
- Others require arguments:
  - **`$ command arg1 arg2 arg3`**
  - entire line is processed
  - command examples:
    - **`mkdir hw1`**
    - **`cp TemplateGUI.java NewGUI.java`**

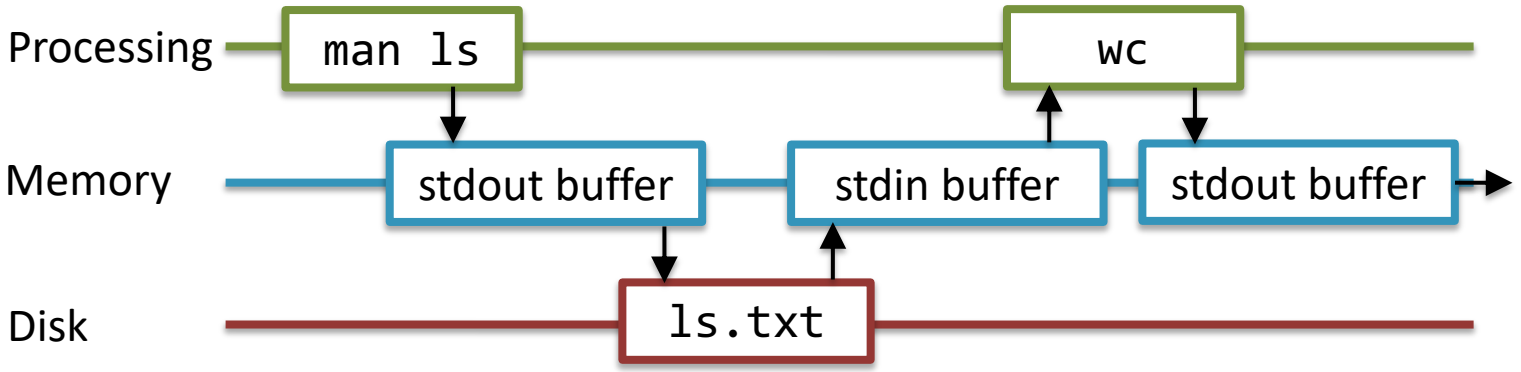
Use the **up-arrow** on your **keyboard** to recall the **last command** that was run!

FILTER COMMANDS

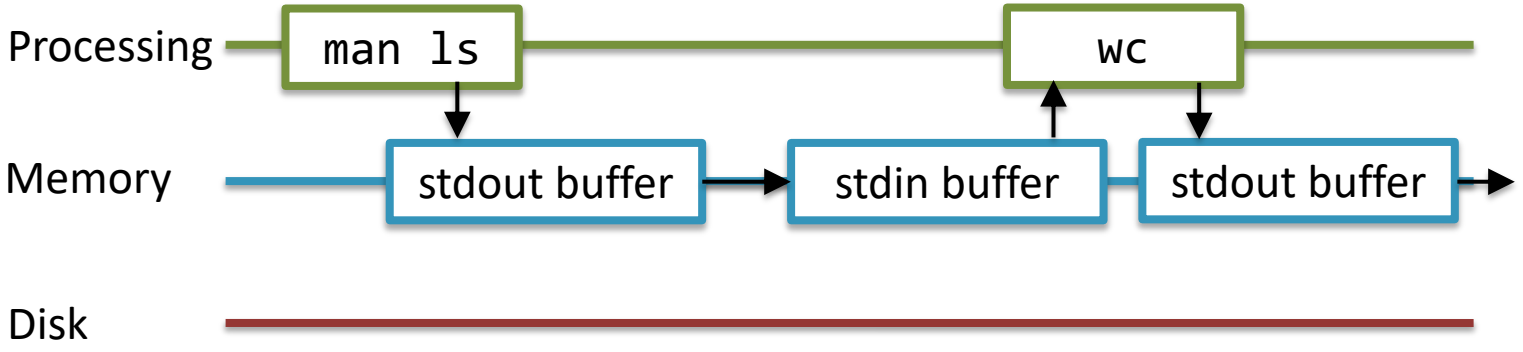
- **grep:** search for a string of characters in a specified file
  - grep: **G**lobal **R**egular **E**xpression **P**rint
  - $ grep <string-to-match> <file-name>
- **more:** displays text one screen at a time
  - $ more [options] <file-name>
- **less:** displays text with more features than **more**
  - $ less [options] <file-name>
- **wc:** print number of lines, words, and characters
  - $ wc [options] <file-name>
- **sort:** sorts lines of text files
  - $ sort [options] <file-name>
- **uniq:** reports/removes repeated lines
  - $ uniq [options] <file-name>

STANDARD IO VS PIPING

`man ls > ls.txt; wc < ls.txt`



`man ls | wc`

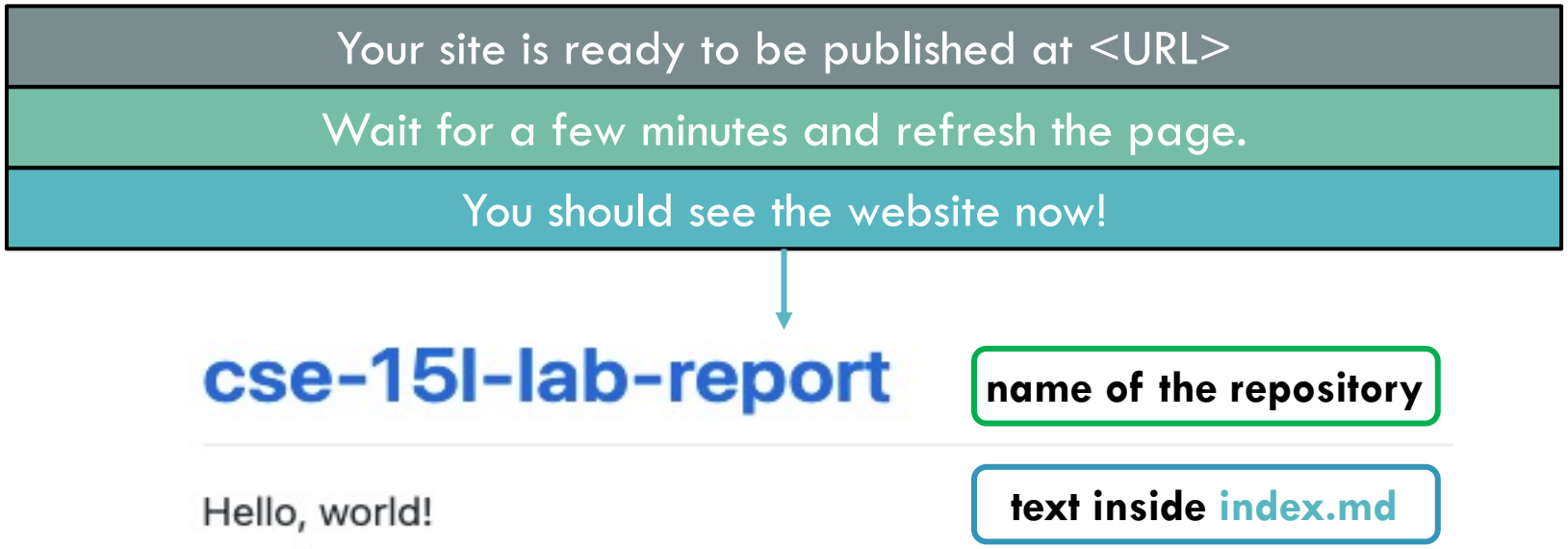


GITHUB

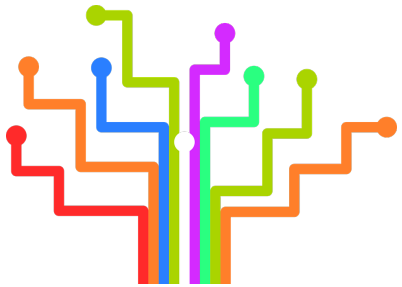
- **GitHub:** web-site for hosting projects that use git.
- **GitHub Repository:** online git repository.
  - analogy to a folder on Google Drive: more control over tracking the history of what changed and by whom.
- Let's go to **https://www.github.com** and create an account.
- Create a new repository
  - Name your repository and click on **"Create Repository"** button.
- Click on **"create a new file"** link.
  - Make a new file called **index.md** and put some text in it.
- Scroll down to the bottom of the page and click **"Commit Changes"**.
- You should be able to see **index.md** file in your repository!

GITHUB PAGES

- **Github Pages:** A service that takes a Github repository and builds a website from it.
- Click on **"Settings"** and then choose **"Pages"** from the sidebar.
- Choose **main** as the source and click **"Save"**.

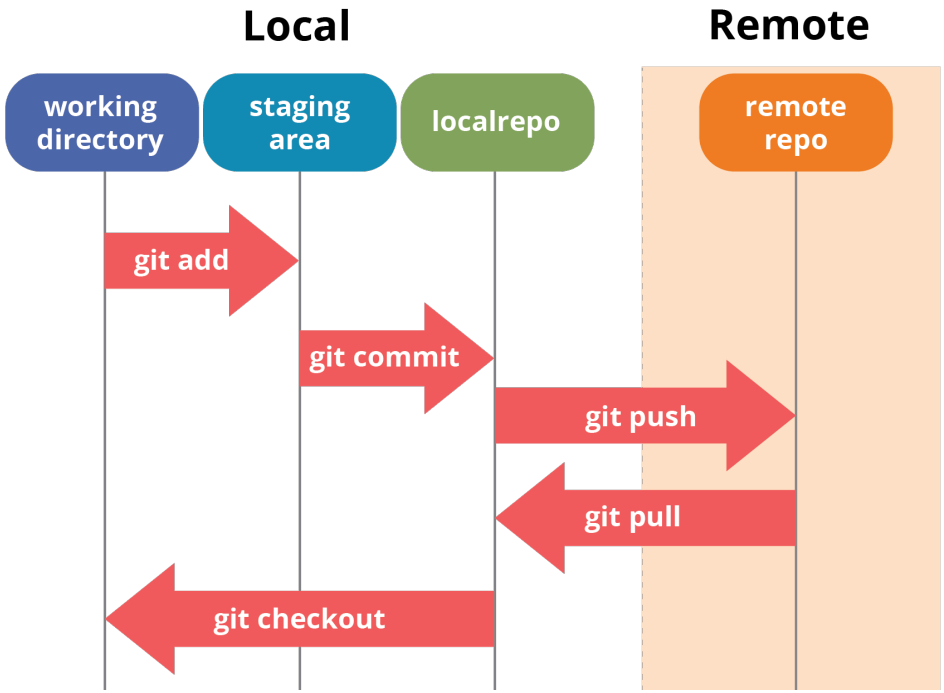


GIT AND GITHUB CONNECTION



- **Let's clone an existing repository on Github and edit this project in the terminal:**

- $ git clone <Github-Project-URL>
- $ echo "some-init-text" > README.md
- $ git add README.md
- $ git commit -m "Create README"
- **push changes to Github**
- **$** git push origin main
- **make some changes on Github and pull them**
- **$** git pull origin main

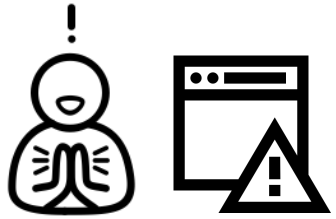


INCREMENTAL DEVELOPMENT

- **Programming strategy** to incrementally write a program's code starting from a few lines, to minimize errors and to avoid debugging long lines of codes.

✓ **Start writing from a short code snippet, add a few lines at each step and keep the code working every time you modify your code.**

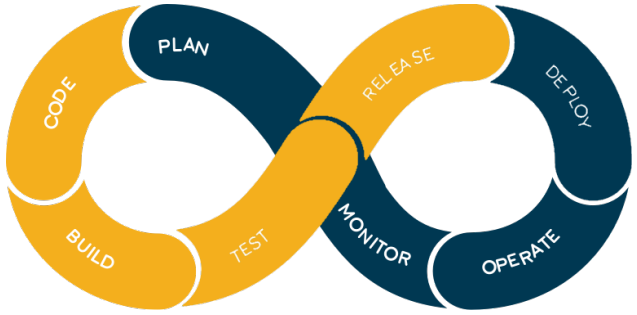
Writing an entire code at once and then testing it all at once. ⊗



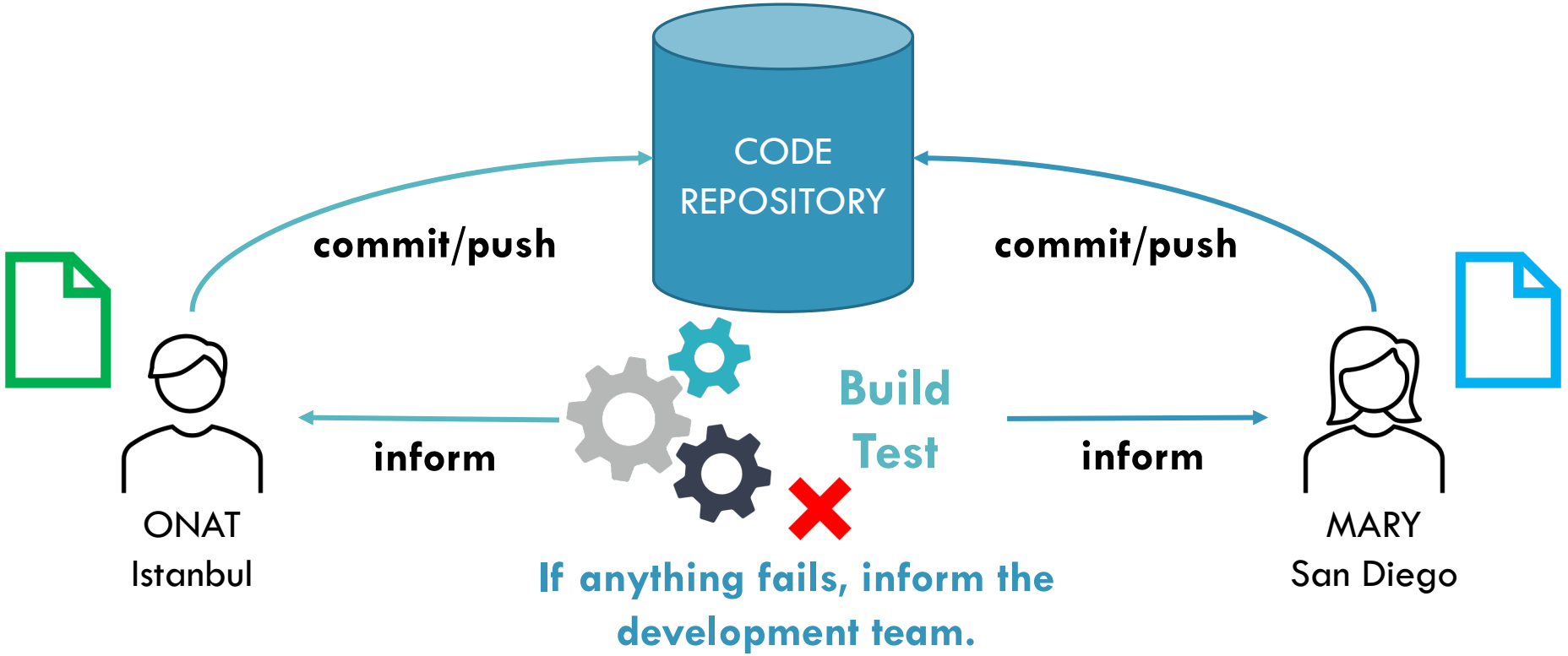
- Advantages of writing code **incrementally:**
  - be aware of flaws in the code earlier
  - figure out which part of the code resulted in a bug since only a few lines is added for each step

- Similar development strategy: **test-driven development**
  - will cover next week!

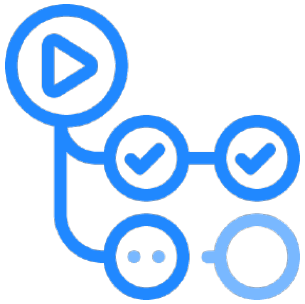
CONTINUOUS INTEGRATION



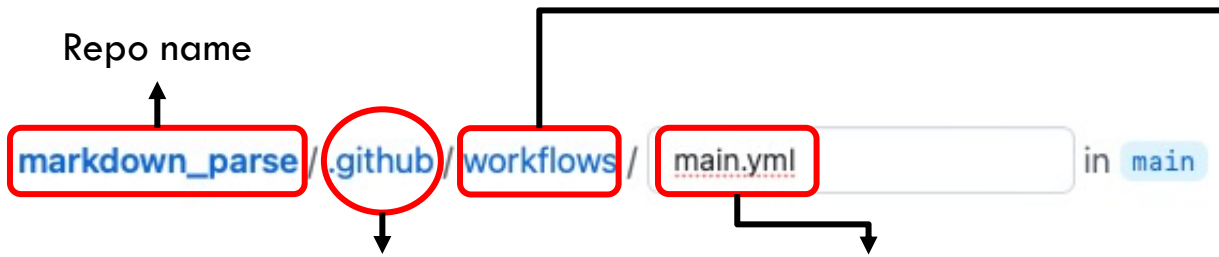
- **Continuous integration:** software development **technique** where developers merge their code changes into a central repository after which **automated builds and tests are run**.



GITHUB ACTIONS



- One of the greatest **tools** available for continuous integration.
  - Other common tools: Jenkins, TeamCity, Bamboo, Buddy, Travis CI.
- Click on Actions in your repository and then select set up a workflow yourself.



Configuration directory for GitHub

YAML = "Yet Another Markup Language" "YAML Ain't Markup Language "

Workflows are defined by a YAML file.

YAML: human-friendly data serialization language for all programming languages

Workflow: configurable **automated process** that will run one or more jobs.

Event: specific **activity** in a repository that **triggers** a workflow run, e.g., pull request, push, issue.

GitHub Actions Events

DEBUGGING

- Some **bad debugging practices**
  - **Making random changes to a code hoping that it would work.**
  - **Googling many things and trying them without understanding what they did.**
  - **If something broke, reverting changes and starting again.**

How many of them did you try?

A) 1
B) 2
C) 3
D) 0

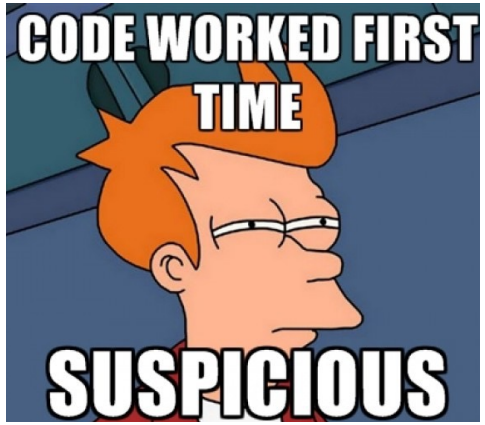
Problematic attitude

This is too hard for me to understand!

Better attitude

Let's learn the basics and see if that helps.

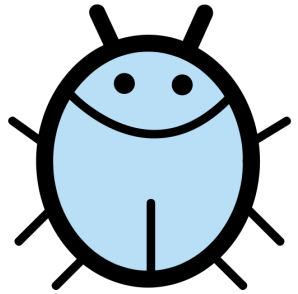
- **How to get better at debugging?**
  - Remember that bugs happen for a **logical reason**
  - **Learn** from the bug
  - **Never** expect your code to work the first time
  - Change **one thing** at a time
  - Read the **documentation**
  - Talk to other **people**



DEBUGGING TERMINOLOGY

- **Symptom:** a faulty program behavior that you can see
  - e.g., crashing or producing the wrong answer
- **Bug:** a flaw in a computer system that may have zero or more symptoms.
  - e.g., syntax errors, semantic errors, warnings, logic errors.
- **Latent Bug:** an asymptomatic bug; it will show itself at an inconvenient time.
- **Debugging:** using symptoms and other information to find a bug.
- **Failure-inducing input:** input to the program that causes the bug to execute and symptoms to appear.
- **Deterministic platform:** A platform having the property that it can reliably reproduce a bug from its failure-inducing input.

THE JAVA DEBUGGER (JDB)



- The java debugger, **jdb**, is a simple **command-line debugger** for Java classes.
- We will use the **terminal** to debug our Java programs using **jdb**.

```
javac -g DebugDemo.java
```

```
jdb DebugDemo <arg>
```

```
stop at DebugDemo.main
```

```
> run
```

```
main[1] locals
```

```
main[1] step
```

```
main[1] cont
```

```
main[1] where
```

```
.suspend
```

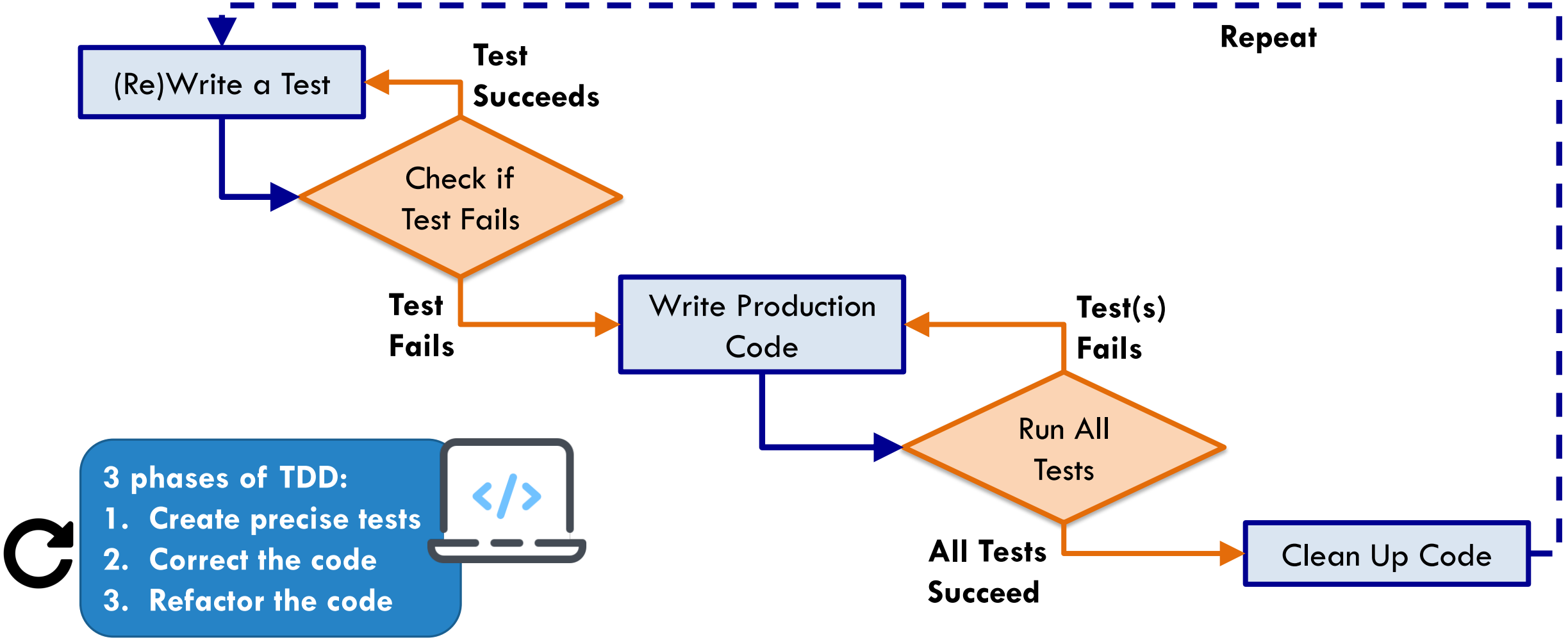
```
threads
```

SYSTEMATIC TESTING

- **Systematic Testing:** test the software against a variety inputs to find bugs
- **One possible idea:** Organize inputs by SIZE
  - **Organize .md files by number of characters**
  - 0-length: “” -> Expected output: [] (empty list)
  - 1-length: “(”, “)”, “[”, “]” -> Expected output: [] (empty list)
  - 2-length: “()”, “[]”, “(]”, “[[” -> Expected output: [] (empty list)
  - 3-length: “[](”, “[()”, “[]]” -> Expected output: [] (empty list)
  - 4-length: “[]()” -> Expected output: [“”] (list w/ single empty link)

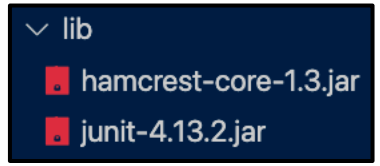
- **Lesson learned:** For testing, organize inputs by size
  - input size: **smallest -> largest**
  - other notions of **size**: number of links, number of lines, number of chars in [] or () etc.
  - great idea to obtain **small failure-inducing inputs**

TDD: TEST DRIVEN DEVELOPMENT



SETTING UP JUNIT

- Download junit and hamcrest .jar files.
  - Java ARchive (JAR) files are essentially .zip files containing Java .class files.
- Make a directory called **lib** in your project and copy both .jar files there.
- Now, let's create our first testing code (MarkdownParseTest.java):




```
import static org.junit.Assert.*;
import org.junit.*;
public class MarkdownParseTest {
    @Test
    public void addition() {
        assertEquals(2, 1 + 1);
    }
}
```

Import required libraries

Run the method as a test case

Check if two inputs are equal

SETTING UP JUNIT

- In order to run this code at the command line, we type the following commands:

```
javac -cp .:lib/junit-4.13.2.jar:lib/hamcrest-core-1.3.jar MarkdownParseTest.java
java -cp .:lib/junit-4.13.2.jar:lib/hamcrest-core-1.3.jar org.junit.runner.JUnitCore MarkdownParseTest
```

```
assertEquals(2, 1 + 1)
```

```
Time: 0.004
OK (1 test)
```

Output 1

```
assertEquals(3, 1 + 1)
```

```
Time: 0.005
There was 1 failure:
...
FAILURES!!!
Tests run: 1,  Failures: 1
```

Oops!

Output 2

SHELL SCRIPTING

Length of a variable:
- **${#<variable-name>}**

run.sh

```
# This script will compile and run MarkdownParse files
CLASSPATH=.:lib/junit-4.13.2.jar:lib/hamcrest-core-1.3.jar
echo Class path is $CLASSPATH
javac -cp $CLASSPATH MarkdownParseTest.java MarkdownParse.java
java -cp $CLASSPATH org.junit.runner.JUnitCore MarkdownParseTest
```

To execute the script: **$ bash run.sh**

Another way to execute: **./run.sh** → You must have the execute permission set to run!

Commenting: Lines starting with # are comments.

Create Variables: <variable-name>=value (NO SPACE!)

Read Variables: $<variable name>

Printing: echo [options] <string-to-print>

To store more complex values, use either single quotes (‘) or double quotes (“).
e.g., myvar=‘Hello World’
myvar=“Hello World”

Variables are strings by default!

MAKE AND MAKEFILES

- **Rules** have the following form:

```
target : dependency1 … dependencyM
    action1
    …
    actionN
```

Use TAB!
Not space

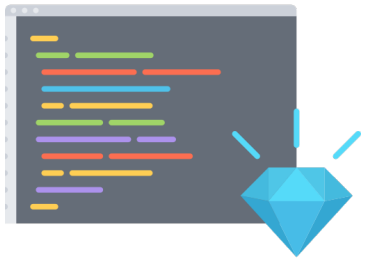
Each line of the script can use standard Unix commands

make compares the **modification time** of the **target** file with the **modification times** of the **dependency** files. Any **dependency** file that has a more recent **modification time** than its **target** file forces the **target** file to be recreated.

- If any dependency is old, **make** will try to re-make it!
- This creates chains of dependence: a file can depend on some files, which depend on other files, etc. **make** can figure all this out!

"to make the **target**,
first make all its **dependencies**,
then perform all the **actions**"

CLEAN CODING



If you write clean code, then you are helping your future self and your co-workers!

"Even bad code can function. But if code isn't ***clean****, it can bring a development organization to its knees. Every year,* ***countless hours and significant resources*** *are lost because of* ***poorly*** *written code. But it doesn't have to be that way."* - Uncle Bob

Why do people write a bad code?	Time pressure, I'm going to fix it later.
How do you know if your code is clean?	Study the principles and practice (a lot).
What is **Clean Code**?	Code that is elegant, efficient, readable, simple, maintainable, and testable.
Why should you care?	Code is (almost) never written once and then forgotten.

UNIX VIM EDITOR

Vim: Vi Improved

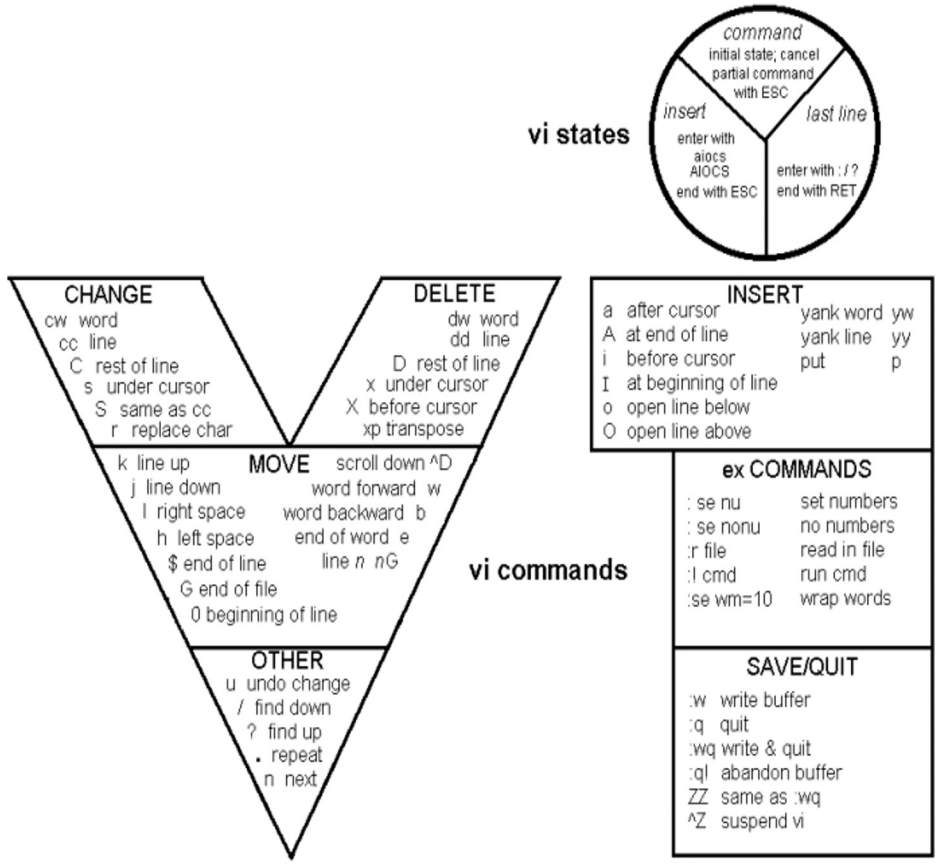
Other Choices: emacs, sublime, nano, …

Good resource: vimgenius.com

- **Vi/Vim** is a free and open-source, screen-based text editor program for Unix.

Why is it useful to know Vim?

- **Lightweight and found on every Unix system**
- **Fast code editing**
- **Commonly used by industry and popular**
- **Very customizable and extensible**



COURSE AND PROFESSOR EVALUATION (CAPE)

https://cape.ucsd.edu

CAPEs began Monday of Week 9 at 8am and will end Saturday of Finals Week at 8am!

If **more than 90%** of students complete CAPEs, then Onat will give **1% extra credit** for all students in the class!

Completion ratio should be **more than 90%** for **all** my sections (B00 and C00)!

Current Status (05/30): 48.98% (B00), 36.36% (C00)

TO DO

Review Today's Lecture Slides

Complete Lab 10

Lab Report 5
(Due: Sunday @ 11:59 pm)