

Programming Skills Workshop

Course Summary

Programmer Skills (Lesson One)

Programming as a Social Practice

In the real world, about 20% of a typical work day is spent coding.

The other 80% is spent on:

- Meetings
- Project Management (Politics!)
- Dealing with Customers
- Training
- Coffee breaks

Programmers don't have to be cool, but they do have to be socially competent.

If you don't work on your social skills, you could be falling short at **80%** of the job!

Skills Checklist - Social

I feel comfortable:

- Interacting in English
- Working in groups
- Participating in class
- Studying (reading in English)
- Writing reports in English
- Asking for help
- Helping/teaching others
- Being a nerd outside of class

Programming as a Technical Practice

Coding may only be 20% of the job (most of the time).

But it's by far the most important part - and the hardest.

Real-life programming requires constant learning and growth.

The skills you'll need to learn programming now are the same you'll be using throughout your entire ICT Engineering career.

Skills Checklist - Technical

I feel comfortable:

- Problem solving
- Administering Windows
- Using the command line
- Understanding programs & tools
- Using Linux (basics)
- Typing (without looking)
- Troubleshooting
- Googling

Learning

“Learning is the **active** acquisition of **skills, knowledge**, and **values** that results in **changed behavior**”

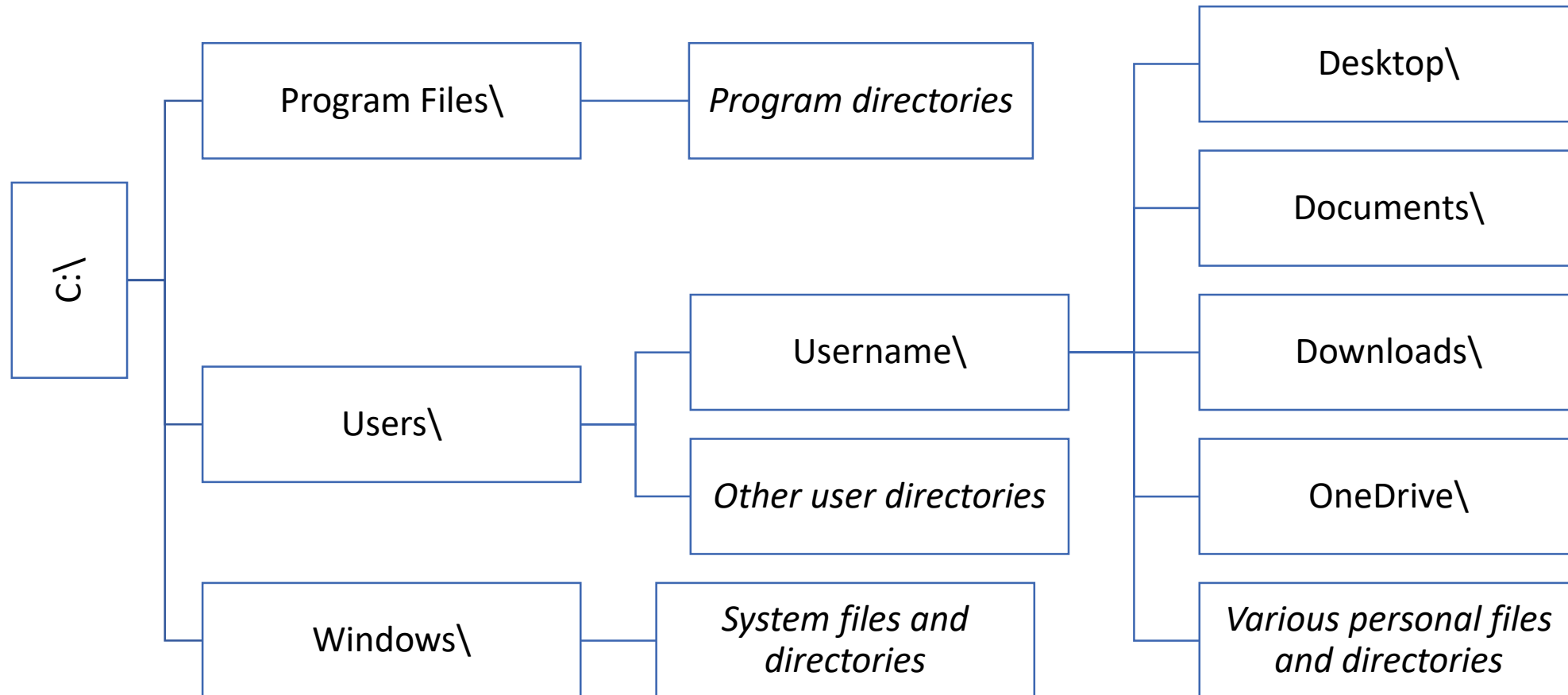
- You won't learn by just being told something (even if you're listening)
- You aren't learning something unless you apply it (changed behavior)

Real, continuous learning is essential to being an ICT Engineer

Windows & Command Line (Lessons Two & Three)

Windows File System Hierarchy

The primary hard drive is usually named C:



Customizing your file system

Common directories for programmers to create:

- **C:\bin**
 - For programs that run on the command line
- **C:\src**
 - For your code

Keep your folders organized!

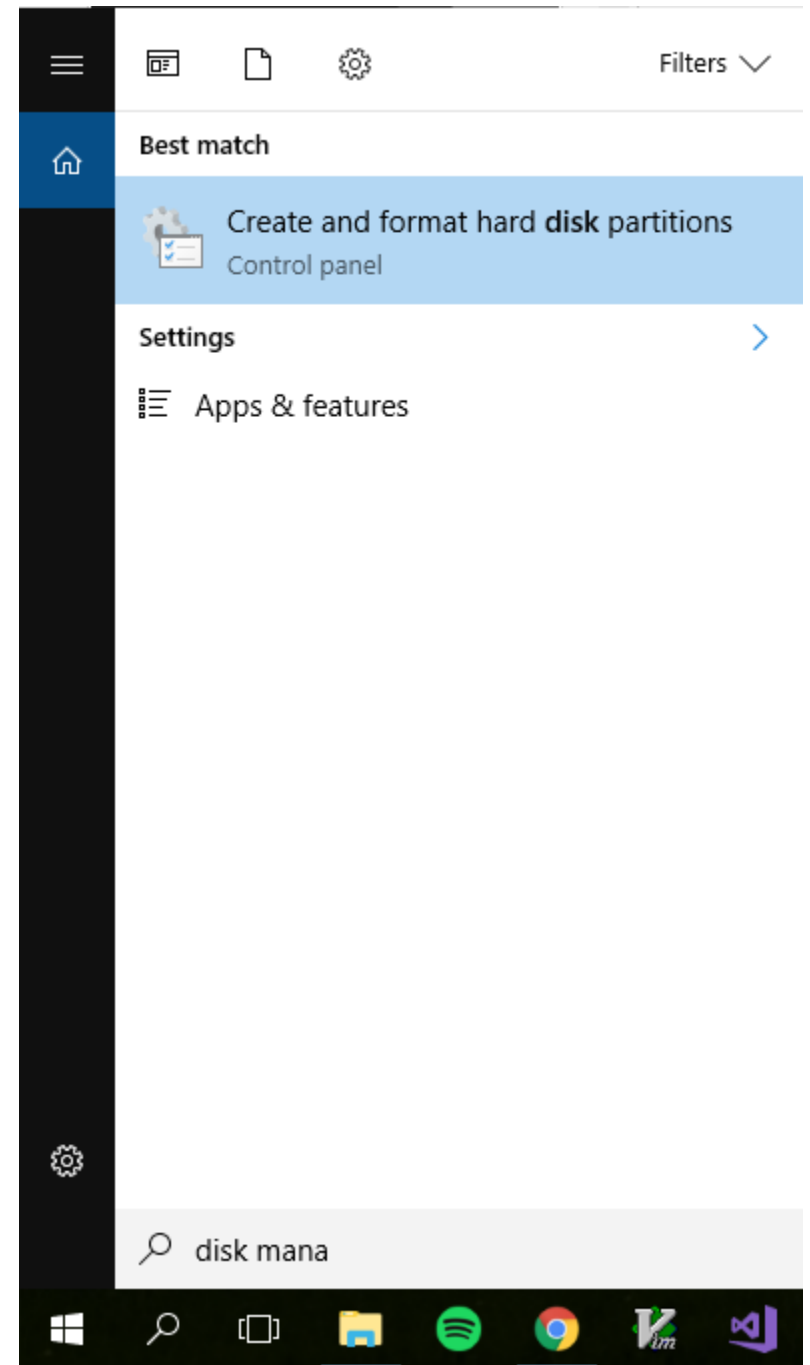
Nothing is more tedious than watching someone stumble through their file system looking for a file ☹️

Finding and Opening Tools

In older versions of Windows, tools were accessed through **Run** or in the **Control Panel**.

In Windows 10, the **search bar** is your friend.

Any Windows tool can be opened by just hitting the Windows key and **typing in the name of the tool**.

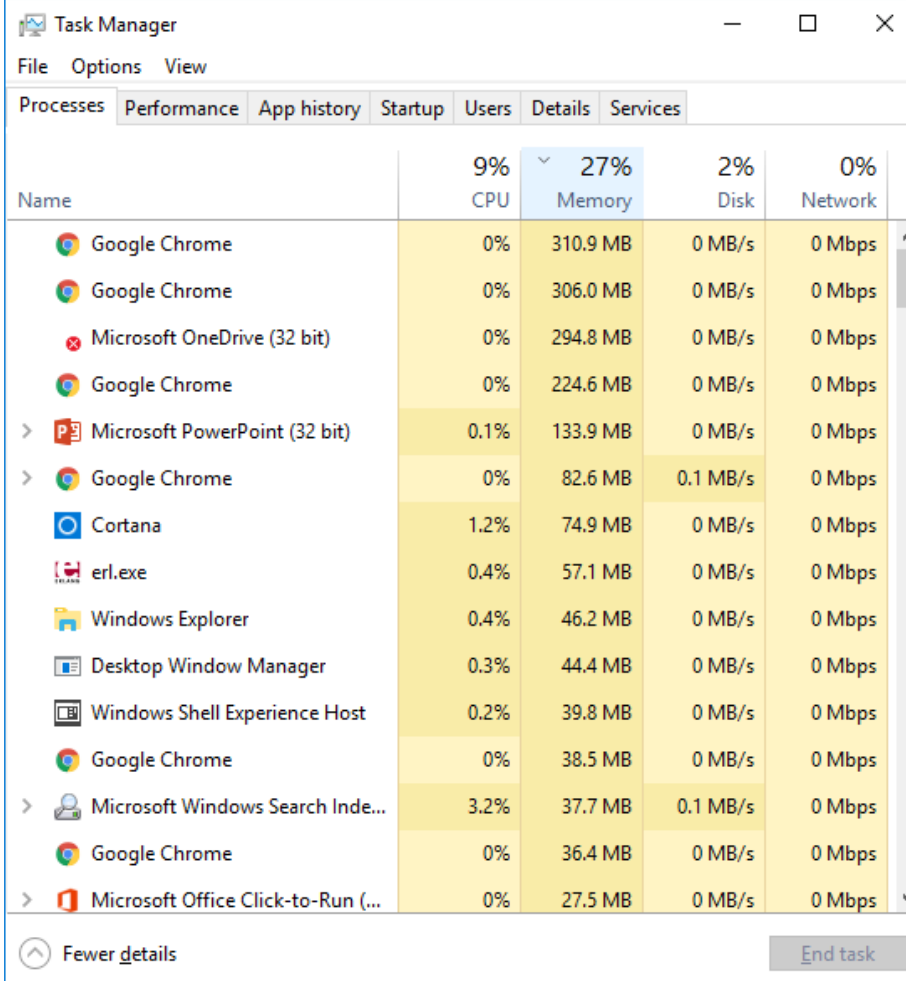


Task Manager

When troubleshooting and debugging
wayward programs,
Task Manager is your best friend.

Task Manager is useful for viewing
performance stats and closing frozen
programs.

In a pinch, you can reach Task
Manager by pressing **Ctrl+Shift+Esc**



The screenshot shows the Windows Task Manager application with the 'Performance' tab selected. The window title is 'Task Manager' and it has standard Windows window controls. Below the title bar is a menu bar with 'File', 'Options', and 'View'. Underneath is a tab bar with 'Processes', 'Performance', 'App history', 'Startup', 'Users', 'Details', and 'Services'. The 'Performance' tab is active, displaying a table of system resource usage. The table has five columns: 'Name', 'CPU', 'Memory', 'Disk', and 'Network'. The 'Memory' column is currently selected, showing a usage of 27%. The table lists various running processes with their respective icons, names, and resource usage percentages and values. At the bottom of the window, there is a 'Fewer details' button and an 'End task' button.

Name	9% CPU	27% Memory	2% Disk	0% Network
Google Chrome	0%	310.9 MB	0 MB/s	0 Mbps
Google Chrome	0%	306.0 MB	0 MB/s	0 Mbps
Microsoft OneDrive (32 bit)	0%	294.8 MB	0 MB/s	0 Mbps
Google Chrome	0%	224.6 MB	0 MB/s	0 Mbps
Microsoft PowerPoint (32 bit)	0.1%	133.9 MB	0 MB/s	0 Mbps
Google Chrome	0%	82.6 MB	0.1 MB/s	0 Mbps
Cortana	1.2%	74.9 MB	0 MB/s	0 Mbps
erl.exe	0.4%	57.1 MB	0 MB/s	0 Mbps
Windows Explorer	0.4%	46.2 MB	0 MB/s	0 Mbps
Desktop Window Manager	0.3%	44.4 MB	0 MB/s	0 Mbps
Windows Shell Experience Host	0.2%	39.8 MB	0 MB/s	0 Mbps
Google Chrome	0%	38.5 MB	0 MB/s	0 Mbps
Microsoft Windows Search Inde...	3.2%	37.7 MB	0.1 MB/s	0 Mbps
Google Chrome	0%	36.4 MB	0 MB/s	0 Mbps
Microsoft Office Click-to-Run (...)	0%	27.5 MB	0 MB/s	0 Mbps

Using the Command Line

Working on the command line couldn't be simpler:

Just write a command, hit enter, then read the response!

You'll see example commands written like this:

```
> command argument1 argument2
```

‘>’ Tells you where you should start writing. In your case, there will be some text before the >. This is your current location.

‘command’ is the actual command to write exactly as shown.

‘argument1’ and ‘argument2’ may or may not be shown, and can (usually) be up to you.

Navigating on the Command Line

Anything you can do in File Explorer can be done on the command line

In Command Prompt:

- List the contents of the current directory:

 > **dir**

- Change to a different directory:

 > **cd** directory-name

Tips – Tab completion

Typing full filenames is a pain in the butt. How can we be lazier?

The **Tab** key is your new best friend! It auto-completes based off the files and folders in your current directory.

Test this! (This time, don't hit enter just yet)

> **cd**

Press tab, and see what happens.

What happens if you type the beginning of a file name, then hit tab?

Tips – Exiting Programs

It's easy to run into frozen or long-running programs in the command line.

Pressing **Ctrl+c** sends an "abort" signal to the currently running program and will return you to the command interface.

Test this!

```
> find "a" war_peace.txt
```

Let's get out of here! Press **Ctrl+c** to return to the console.

The PATH

The **PATH** is a simple variable that tells Windows where to look to find programs that run on the command line.

To check your current PATH, type:

> **path**

To change the PATH, go to :

- System -> Advanced System Settings -> Environment Variables
- In the **System variables** box, click on **Path** and then select **Edit...**
 - *Note that you must separate directory paths by using a ; character*

Using Command Line Programs

To run a program on the command line, it either needs to be in the current directory or listed in the PATH variable.

In both cases, running a program is as simple as typing the name of the program, followed by any options or arguments that it might need.

Examples:

```
> ping www.google.com
```

```
> ipconfig
```

Command line Java

You don't need to use Eclipse to compile and run your Java programs!

The programs you want to use are simply called **javac** and **java**.

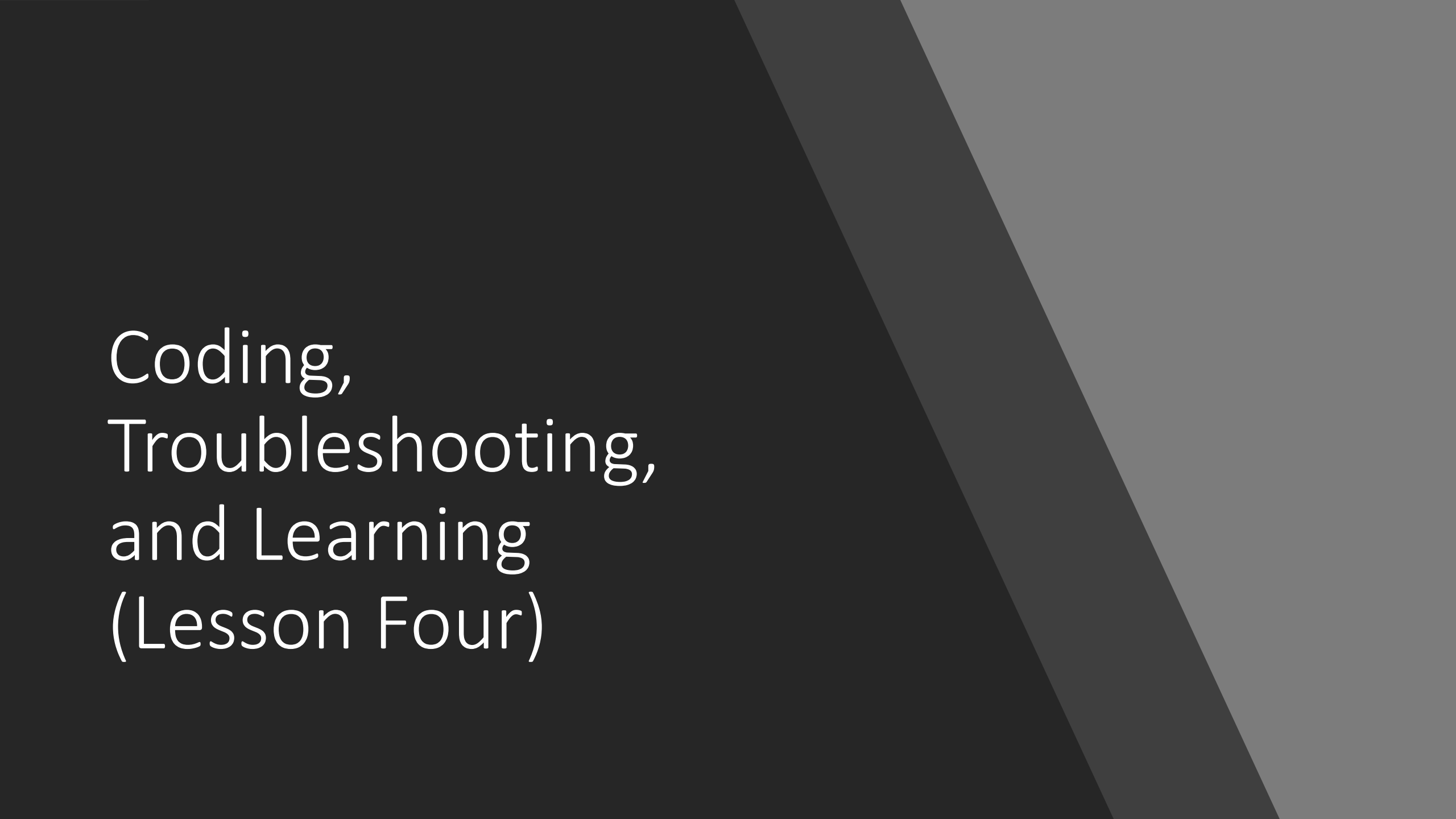
If these programs are included in your **PATH** variable, you can use them from your command line.

1. To compile a **.java** file:

1. Create a file containing a valid Java program.
2. Run **javac** on the file. The Java compiler will create a new **.class** file.

2. To run a compiled Java program:

1. Run **java** on the **.class** file. Be sure to omit the **.class** extension.



Coding, Troubleshooting, and Learning (Lesson Four)

Clean Code

*“Any fool can write code that a computer can understand.
Good programmers write code that humans can understand”*

- Martin Fowler

Code is meant to be read and understood by humans.

For this reason alone, it is important to keep you code **clean** and **organized**.

Naming

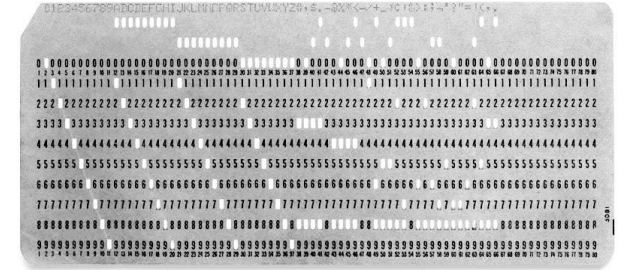
“There are only two hard things in Computer Science: cache invalidation and naming things.”

- Phil Karlton

Handy rules for naming things:

- Follow **conventions** for your language/platform (Google it!)
- Avoid repetition (String nameString -> String name)
- Use **full words**, not single letters or abbreviations
 - Unless it's convention, like **i** for iterators and **T** for generics
- When in doubt, **clarity** wins over simplicity
 - Ask “does this name **fully describe** what this does?”

Code Length



Longer code is harder to understand.
Code should be kept short and sweet.

Follow **principles** instead of worrying about exact numbers:

- Classes and methods should **do one thing**
- **Don't Repeat Yourself**
- Don't write code that you “might” need eventually
- Separate **high-level** and **low-level** code with **abstractions**

Helpful Resources

Codecademy – Guided coding lessons

www.codecademy.com/learn/learn-java

Derik Banas – The only good Java tutorial videos

www.youtube.com/watch?v=TBWX97e1E9g&list=PLE7E8B7F4856C9B19

Computer Science Crash Course – Great videos for explaining programming concepts

www.youtube.com/watch?v=tplctyqH29Q&list=PL8dPuuaLjXtNIUrzyH5r6jN9ullgZBpdo

The Impostor Syndrome

Since programming is difficult, you'll find that **every programmer struggles with confidence from time to time.**

Even expert programmers with decades of experience can feel like they aren't good enough, and that they don't belong. Even when they're writing code, they feel like at any time their peers will "reveal them" to be a fraud.

This lack of confidence is so common that it has a name – **the Impostor Syndrome.**

The best remedies for this are (once more) **persistence** and **curiosity.**

Asking Questions

Many IT professionals, especially software developers, are afraid to ask questions, in fear of exposing gaps in their knowledge.

This is a terrible mistake.

Programmers are always facing new challenges, and nobody can possibly be expected to know everything.

All programmers get lost and feel stupid from time to time – the best know when and how to get help!

Asking good questions will help you keep pace, and shows others that you are interested, disciplined and motivated.

Solving Problems

Programming is essentially applied problem solving.

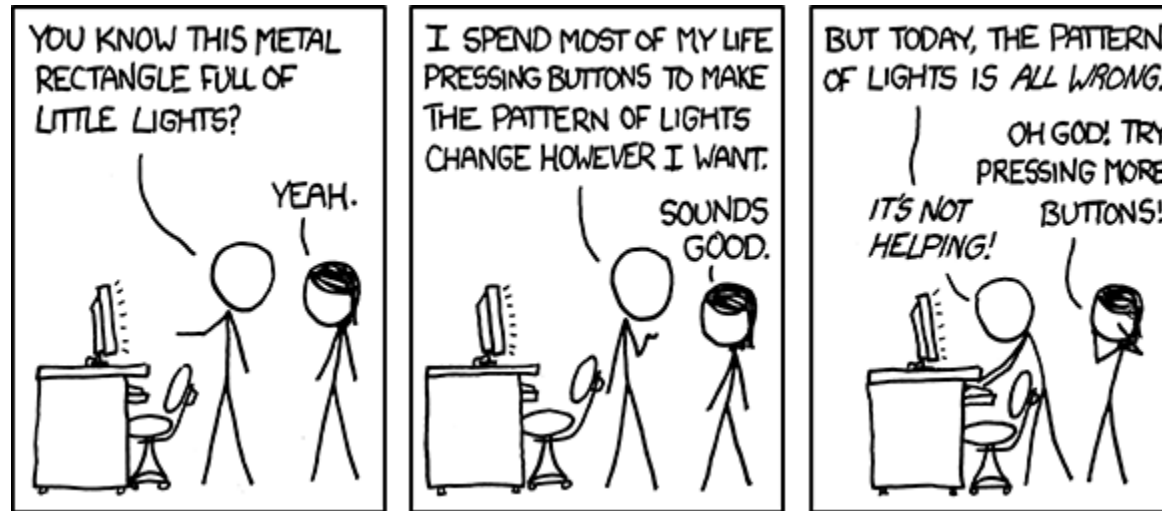
It is not enough to rely on hacking around or following tutorials.

To be a good ICT Engineer, you will have to learn how apply a process:

1. Understand the Problem
 - Break it into pieces if you need to
2. Devise a Plan
3. Carry out the Plan
4. Review

Troubleshooting

As an IT student, you should be able to work through most computer-related problems on your own.

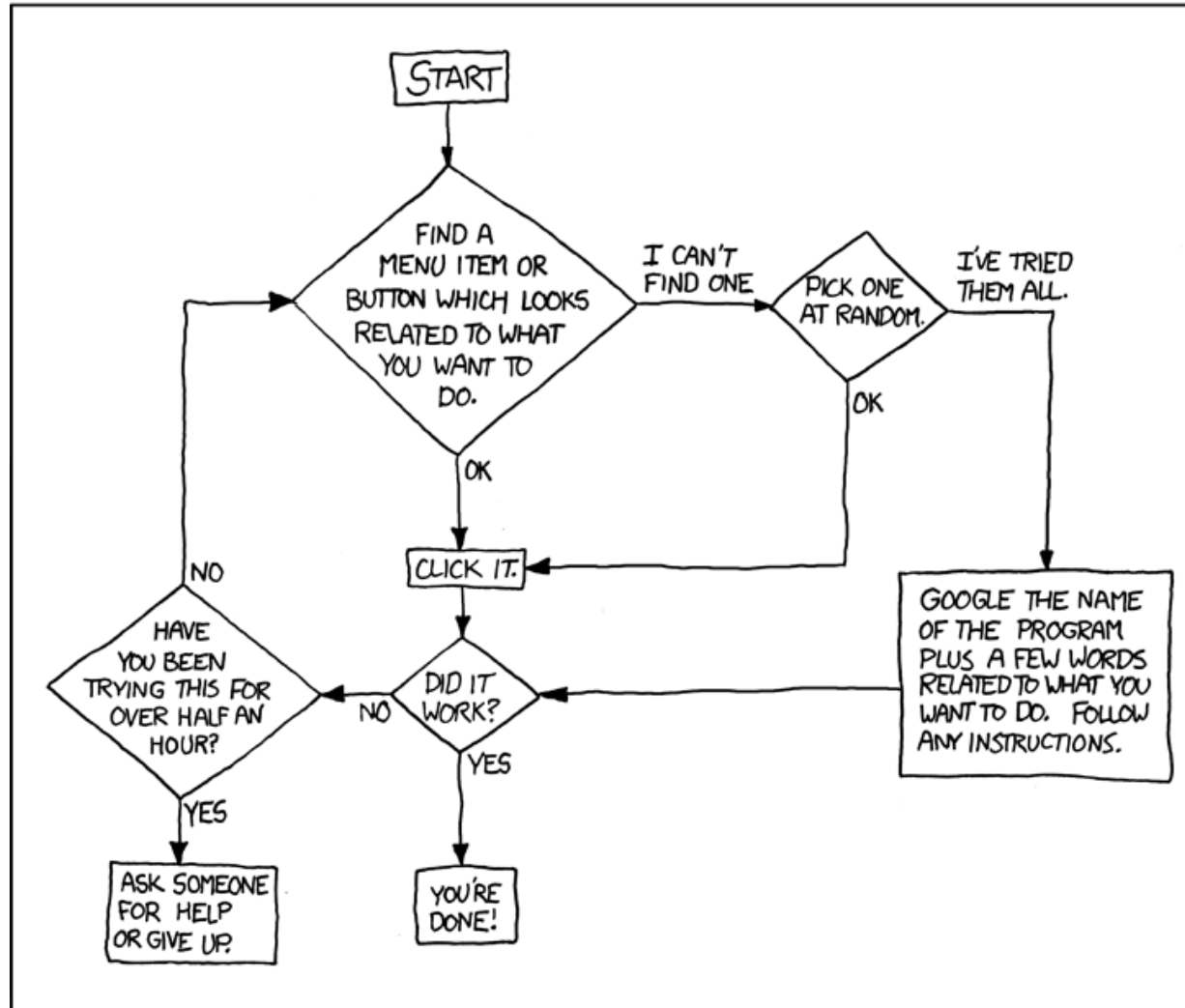


Source:
<http://xkcd.com/722/>

The Secret of IT Troubleshooting:

You'll almost never know the answer to a problem before it comes up.
Instead, a simple algorithm can be used to solve virtually any IT problem.

Troubleshooting Cheat Sheet



Source:

<http://xkcd.com/627/>

Using high-quality sources

Not all sources of information are equal!

The Reference Hierarchy:

1. Official documentation and source code ([Oracle Java API](#), [MSDN docs](#))
2. Up-to-date books
3. Peer-reviewed documentation ([Mozilla Developer Network](#))
4. Blog posts by expert professionals ([Microsoft MVPs](#), [CSS Tricks](#), etc.)
5. [Stackoverflow](#) (most of the time)
6. Forum threads (on reputable sites)
7. Blog posts on personal (non-ad) sites

Avoiding low-quality sources

Not all sources of information are equal!

Poor sources include:

- Ad-based "tutorial" sites (W3Schools, About.com, WikiHow, etc.)
 - Make sure you can see **who** wrote the content, **when** they wrote it, and what their **qualifications** are!
- Out-of-date books and documentation
- Anything on a site that has too many ads, is outdated, or just plain looks bad



Source Quality Checklist



For every source, ask:

- **Who** wrote this? Why should you trust them?
- **When** was this written?
 - Is the information provided still relevant?
- **Why** was this written?
- What **sources** are provided? Are they relevant and up-to-date?
 - Apply the same process to all major sources!
- Who paid for this to exist? Why?
 - Are there **ads**? Do they distract from the content?

Textbooks – The Skimming Method

Textbooks exist for a reason – they are the first place you should go to learn more about topics that you couldn't fully understand during course lectures.

However, **textbooks should not be read completely**, from top to bottom!
Textbooks contain more details than you'll ever need or be able to remember.

Learn to **skim** textbooks for the information you actually need:

1. Find the relevant section (using the Table of Contents or Index).
2. Read completely the opening and closing paragraphs in the section.
3. Read the first sentence(s) in each paragraph.
 - Write a **brief** summary of each in **your own words**.
4. If you still don't understand something, re-read the relevant paragraphs **completely** and write a **detailed** summary of each.

Text Editors & IDEs (Lesson Five)

Text Editors vs. IDEs



The humble **text editor** is a developer's best friend. Try them all, find one you like the most, and configure it to your heart's desire.

An **IDE**, on the other hand, will often be made for a specific language or platform. A .NET developer, for example, will use Visual Studio for most coding tasks.

Text editors can be used to edit all types of text files, but IDEs should be used for their specific purpose. It is common to use a text editor and IDE together in a project.

Make sure you are using the right tool for the job!

What a Text Editor isn't

A text editor is not an IDE, even if it can “become” like one

- Text editors are for single files and short scripts
- IDEs are for projects

For any platform there is usually a better IDE

- Text editors can be great **companions** to IDEs

You ***really*** don't want to use Brackets for Java!



Eclipse Autoformatting

Do you like to type sloppy and clean up later? Eclipse can help!

Type **Ctrl + Shift + F** to have Eclipse autoformat (“pretty print”) your code.

In **Window -> Preferences -> Java -> Code Style -> Formatter** you can change and add code styles

- Google’s style guide (github.com/google/styleguide/blob/gh-pages/eclipse-java-google-style.xml) is a good standard to follow.

If you want to autoformat code in a whole project, you can right-click on the project and select **Source -> Format**

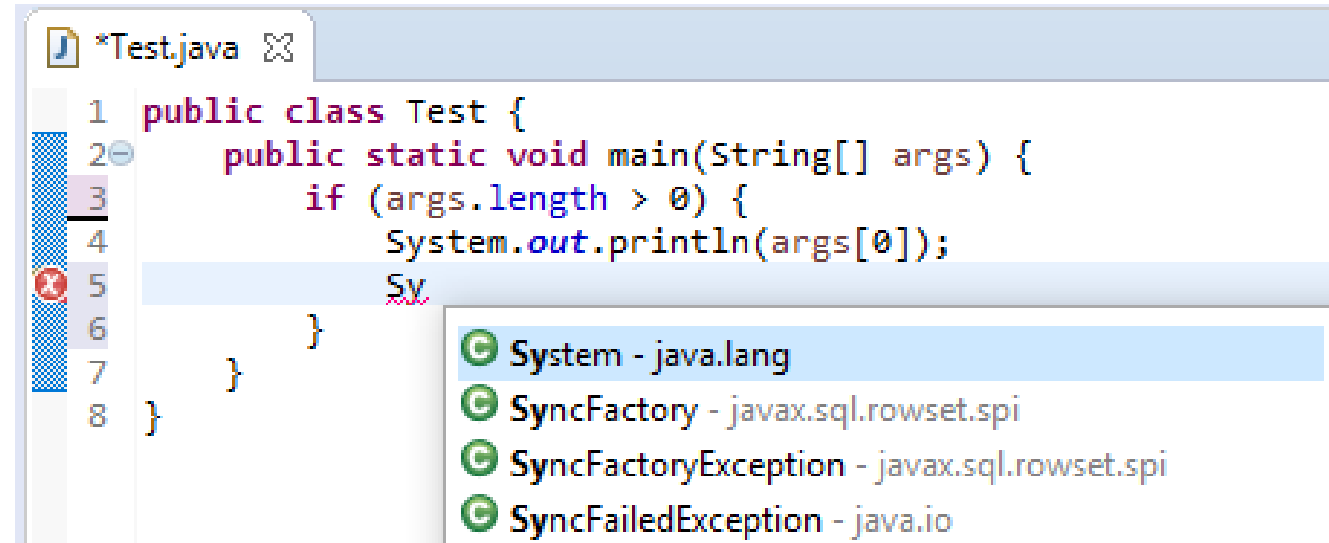
Eclipse Autocomplete

Be lazy!

Don't type long words manually!

Using **Ctrl + Space**, Eclipse will finish words for you.

This is a great way to write easily-misspelled class and variable names.

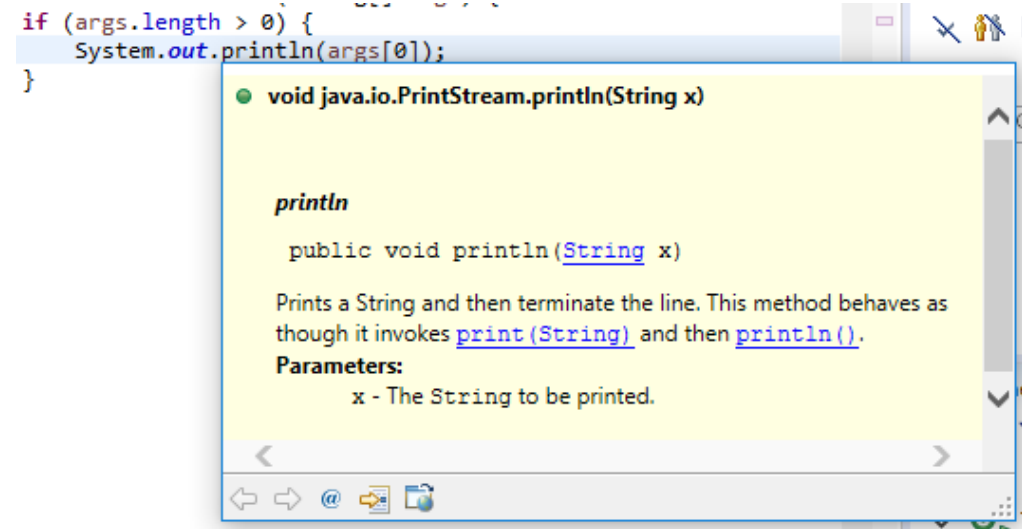


Eclipse Code Completion

Code Completion (“intellisense”) is a common feature in IDEs.
It is your **new best friend**.

Eclipse tells you what a class or method does - without you having to look it up in documentation!

To activate this feature, simply hover your mouse over a class, method or variable.



Javadoc comments combined with code completion make it easy to code with Java libraries!

Using a debugger

Print statements are a very primitive technique, and aren't interactive.

A modern debugger will **literally pause** your program while it is running, and let you view and even change the current values.

Using a debugger, you can **step through** code **one line at a time** to see exactly where bugs occur.

A debugger is not magic! It just feels like it.

Debugging in Eclipse

Tools like the Eclipse debugger make debugging an intuitive process.

To open the debugger, choose **Run -> Debug**, hit **F11**, or use the debug perspective button.

Eclipse has a weird “perspective mode” for its debugger.

When done debugging, you can switch back to normal mode with the perspectives button in the upper-right of the program.



Debugging Process

The normal debugging process is simple (and the same in most IDEs):

1. Place **breakpoints** in your code
2. Run the program in **debug mode**
3. The program will pause at each breakpoint
 - Use the **inspector** to view the values of variables
 - Use **Step Into** or **Step Over** to advance to the next line (and repeat step 3).
 - **Step Into** will enter into any method that is called on the current line
 - **Step Over** will skip over any method that is called on the current line
 - Use **Resume** to continue your program (until the next breakpoint).

Alternatively, you can pause a running program at any time.

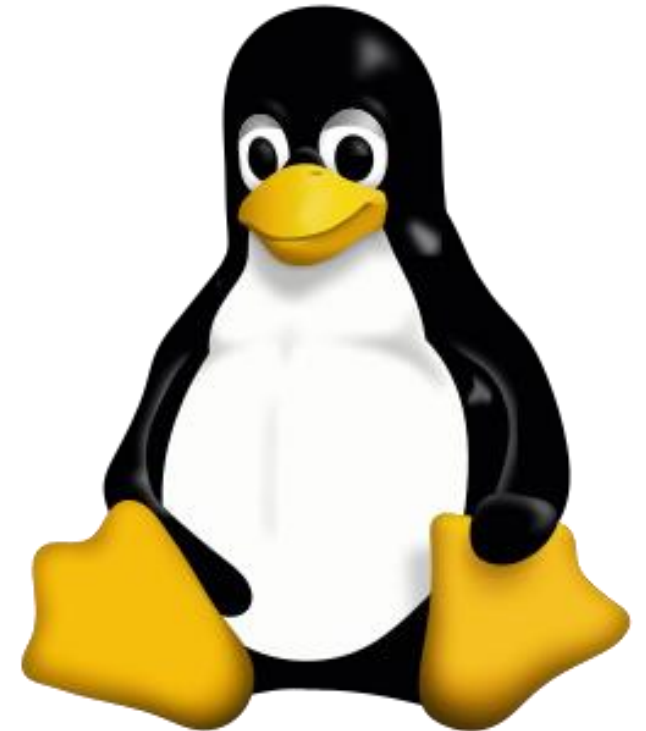
Linux Background (Lesson Six)

Linux in review

Linux is the common name for any operating system built on top of the Linux kernel.

Linux is often used in a **command line interface**.

Any Linux installation should come with a large number of tools for installing programs, configuring the operating system, and working with code.



Much ado about licenses

Licensing is a **big deal** in software.

A software license allows software to be legally used and distributed.

Billions of dollars are made by selling licenses, and billions of dollars are won in legal courts over software license disagreements.

Source code is also a big deal. Most software is **closed source** (secret).

Open source software (OSS) licenses allow for the source code to be read and changed (depending on license). OSS can be sold or given away for free.

The big problem with Unix, in the beginning, was its unclear licensing rules. Nobody wants to touch software with unclear licenses.

Navigating on the Linux Terminal

> **cd [target]**

Changes directory to the target.

If no target provided, goes to your user home directory.

> **pwd**

Tells you where you are.

> **ls**

Lists all folders and files in your current directory.

Same as **dir** from windows.

File manipulation on the Linux Terminal

To create a directory:

```
> mkdir [directoryName]
```

To remove a file:

```
> rm [fileName]
```

To remove a directory and all its files (dangerous!!!):

```
> rm -rf [directoryName]
```

Linux File Structure

/ - is the root directory of the file system

~ - is your home directory. Its an alias for /home/<username>

To see the full hierarchy run **man hier** or read wikipedia page about **FHS**

- Interesting files

- /dev/null – everything written here is immediately lost
- /bin – contains all programs used by user or system during run
- /tmp – temporary directory that may be randomly erased
- /media – this is where devices get mounted (USB, DVD,...)

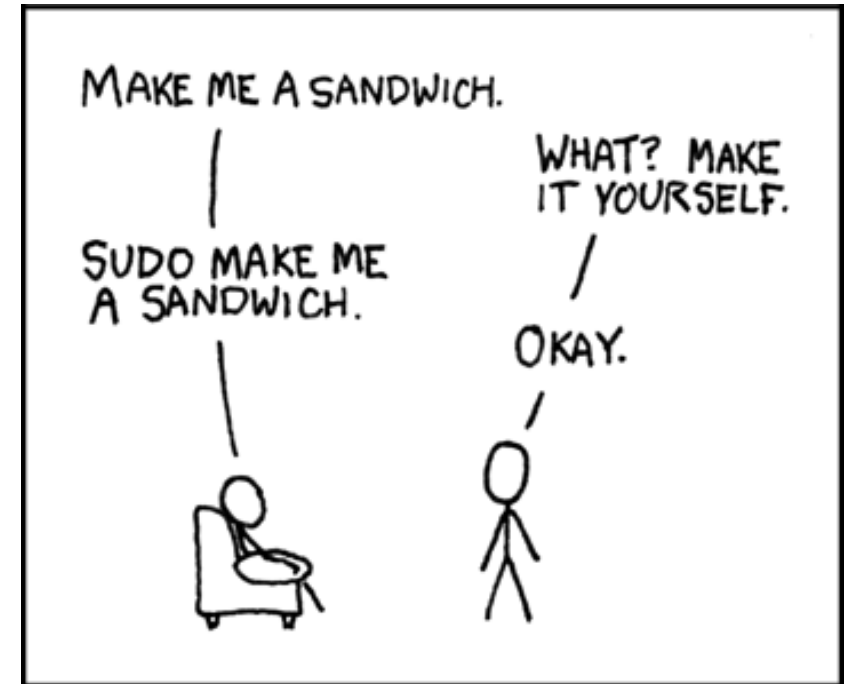
Root & Privileges

Root is a Unix term for SuperUser.
(Same as administrator in Windows).

Root can do **anything** and change anything on a computer (including breaking it!)

To execute program as root you can:

- Use the **sudo** prefix before a command:
 > **sudo** [command]
- Login with the **root** account



Programming Languages (Lesson Seven)

Why are there so many?

The only correct answer to “what language is best?” is “**it depends!**”

A programming language is a tool for solving problems.
Each language offers a different approach.

Often, languages are better at solving certain problems:

- How do I access data in a database? -> SQL
- How do I manage payroll for my employees? -> Excel
- How do I make sure my rover works on Mars? -> C

The language you choose should fit the problem you want to solve.

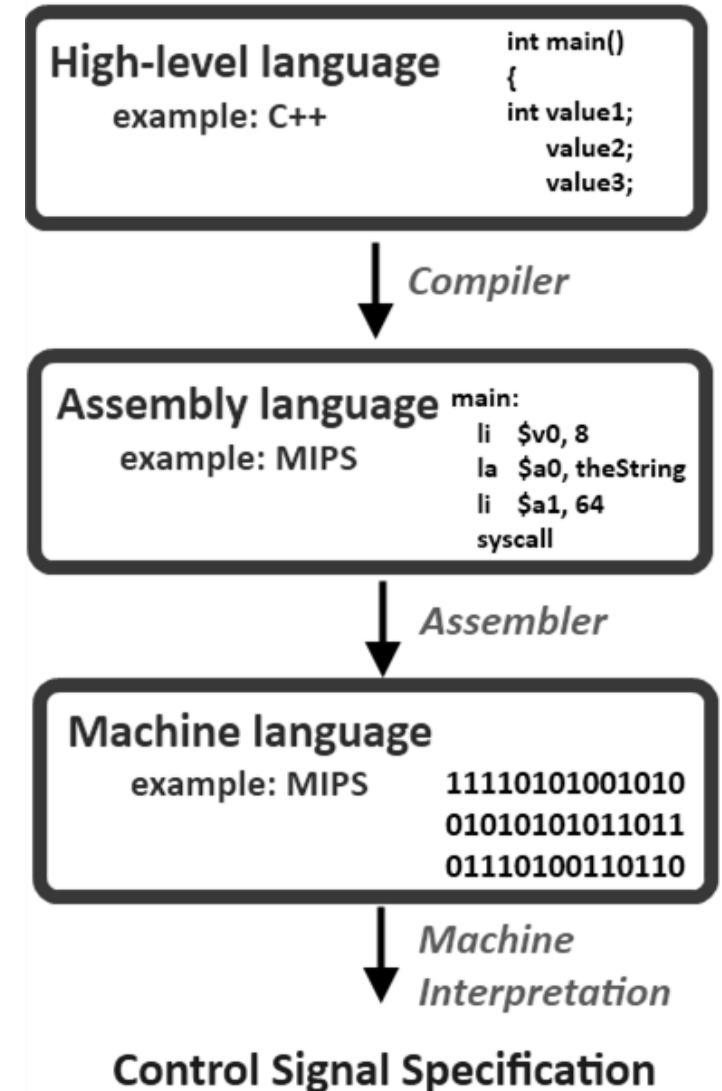
Levels of Abstraction

A core concept in programming is **abstraction**.

“The essence of abstractions is **preserving information that is relevant** in a given context, and **forgetting information that is irrelevant** in that context.”

– *John V. Guttag*

Modern programming languages tend to offer high levels of abstraction.



Programming Paradigms

All languages can be (roughly) divided into a set of **paradigms**.

A paradigm is a way of organizing code.

- Usually to take a specific approach to solving problems.

Depending on the problem you need to solve, different paradigms will offer different abstractions to help you.

Imperative and Structured Programming

In an **imperative** paradigm, a program is a set of **commands** that are followed top to bottom, until none are left.

- Think of this like a simple recipe.
- This is the most primitive way to organize code.
- Most programming languages support imperative programming.

In **structured programming**, imperative code is given structure by things like **loops** (while/for), and **conditional statements** (if/else, switch)

Non-structured programs rely on “jumps” to line numbers in code

- This is “considered harmful”

Compilation

For code written in a high-level language to run on a computer, it must be **translated** into machine code instructions.

This is traditionally done using one of two programs:

- A **compiler**, that translates the code all at once, so it can be run later.
- An **interpreter**, that translates and runs the code one line at a time.

Multi-Paradigm

We have seen a few programming paradigms so far:

Imperative: Program is a series of statements, run one-by-one.

Functional: Program is a series of evaluating functions.

Object-Oriented: Data and functionality are paired together in objects.

Modern languages like Java are considered **multi-paradigm** because they contain elements of multiple paradigms at once.

Java



The first popular **object-oriented, garbage-collected** language.

Java was released in 1995 by **Sun** (now owned by **Oracle**).

Java looks like C/C++, but is a higher-level language.

Java's killer two features were:

1. **Portability - “Write once, run anywhere”**

- The same Java code can be run on a huge number of platforms.

2. **Easy OOP**

- Java is object-oriented with classes, and offers automatic garbage collection.

The Million Dollar Question

What programming language is best for teaching?

C/C++ for classic programming (Too hard!)

Python for modern programming (Too unfamiliar!)

C# for enterprise programming (Too Microsoft!)

JavaScript for web/hipsters (Too JavaScript!)

Java

The winner is...Java!

Java is:

- **Widespread**
 - On billions of devices, from embedded to Android to big data.
- **Mature**
 - 20+ years as a top-three language.
- **Robust**
 - The Java SDK libraries are sprawling and easy to use.
 - Core programming principles are easy to learn in Java.
- **Beginner-friendly**
 - Learning to program with Java is a smooth, gradual process.
 - Especially with tools like Eclipse.
- **Scalable**
 - Java works equally well for simple class exercises and gigantic open-source projects.





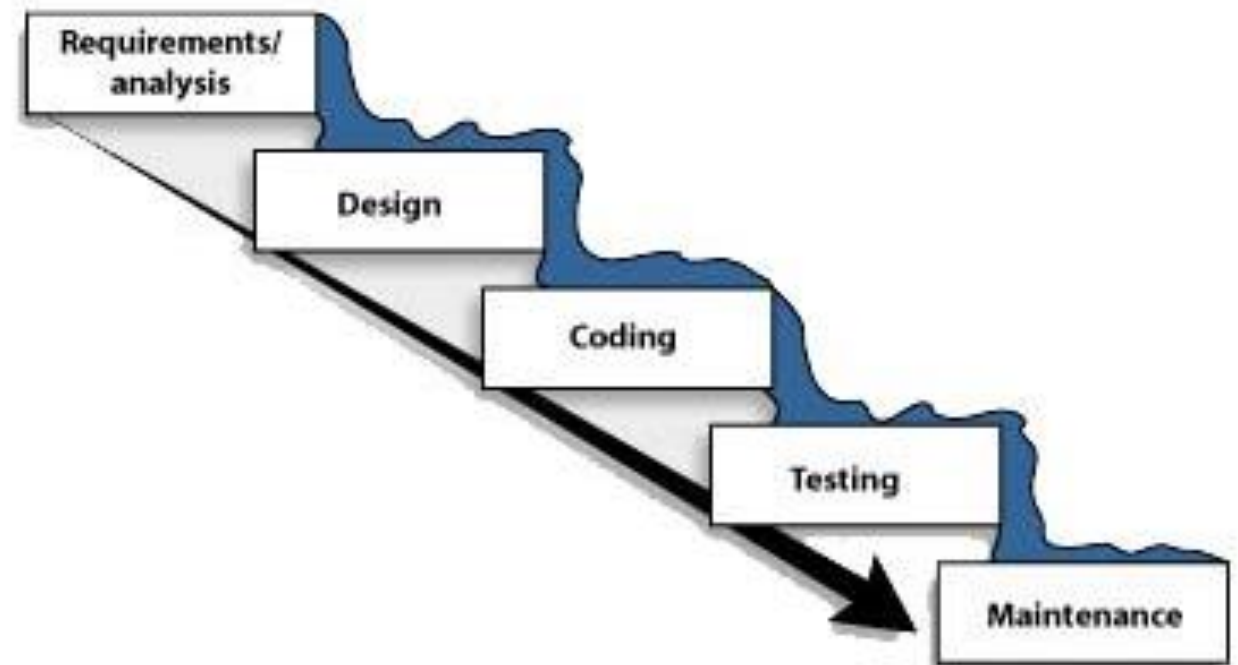
Team Work Processes & Tools (Lesson Eight)

The Waterfall Method

Waterfall is the standard and most common software development process.

The following activities happen in order:

1. **Requirements**
 - Find out what to code
2. **Analysis**
 - Find out what is needed to code it
3. **Design**
 - Plan how to code it
4. **Implementation**
 - Code it
5. **Testing**
 - See if the code works
6. **Maintenance**
 - Continue fixing the code forever



Looks great!

Agile

Agile is a family of methodologies and processes.

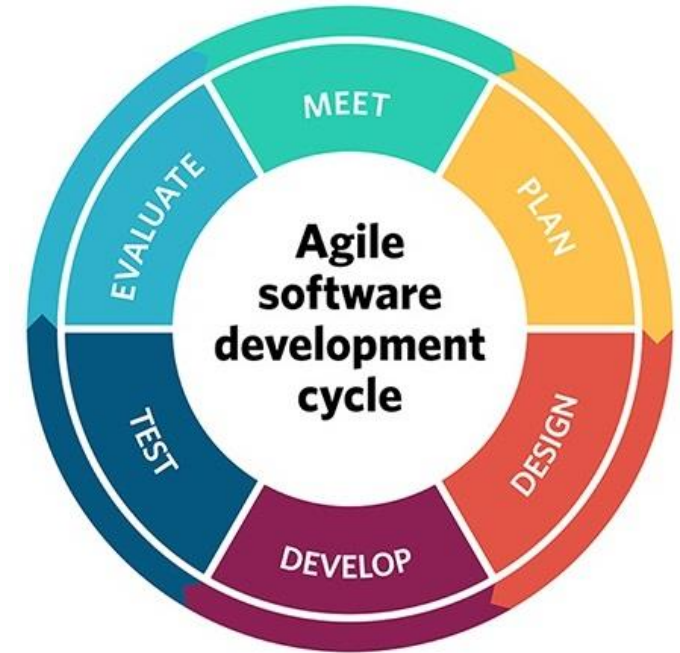
- A **set of beliefs and values** instead of hard rules.

Agile processes tend to be **iterative**:

- Design, Build, Test, Repeat.
- Change is **embraced**, even late in product development.
 - We (and our customers) never know what we want at the start
 - What was valuable yesterday may not be valuable anymore

The two most used Agile methodologies right now are: **Scrum** and **Kanban**

- Scrum is about following a process and using specific tools
- Kanban is about visualizing work



What is Git?



Git is a **distributed version control system**.

- In a distributed system every developer keeps their own copy of the code.
- In a centralized system the code is stored in one central place.

Referred to by its creator, Linus Torvalds, as a “stupid content tracker”.

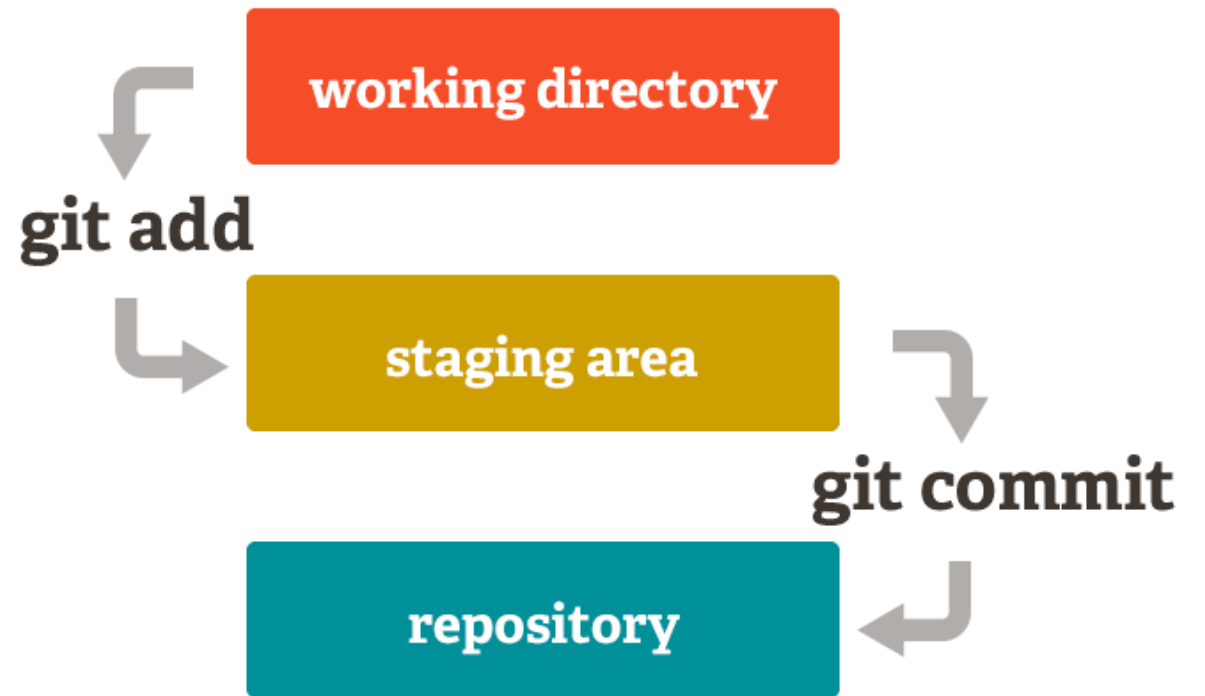
- Created in 2005 to help develop the Linux kernel.
- Nothing else was fast enough or suitable for distributed teams.
- It is extremely powerful, but the core features are simple.

Git is by far the most popular VCS tool today.

It is free, and open source under the GNU General Public License.

Basic terms

- Repository
- Staging area
- Commit
- Remote
- Branch
- Collaborator



Git Commands

`git init`

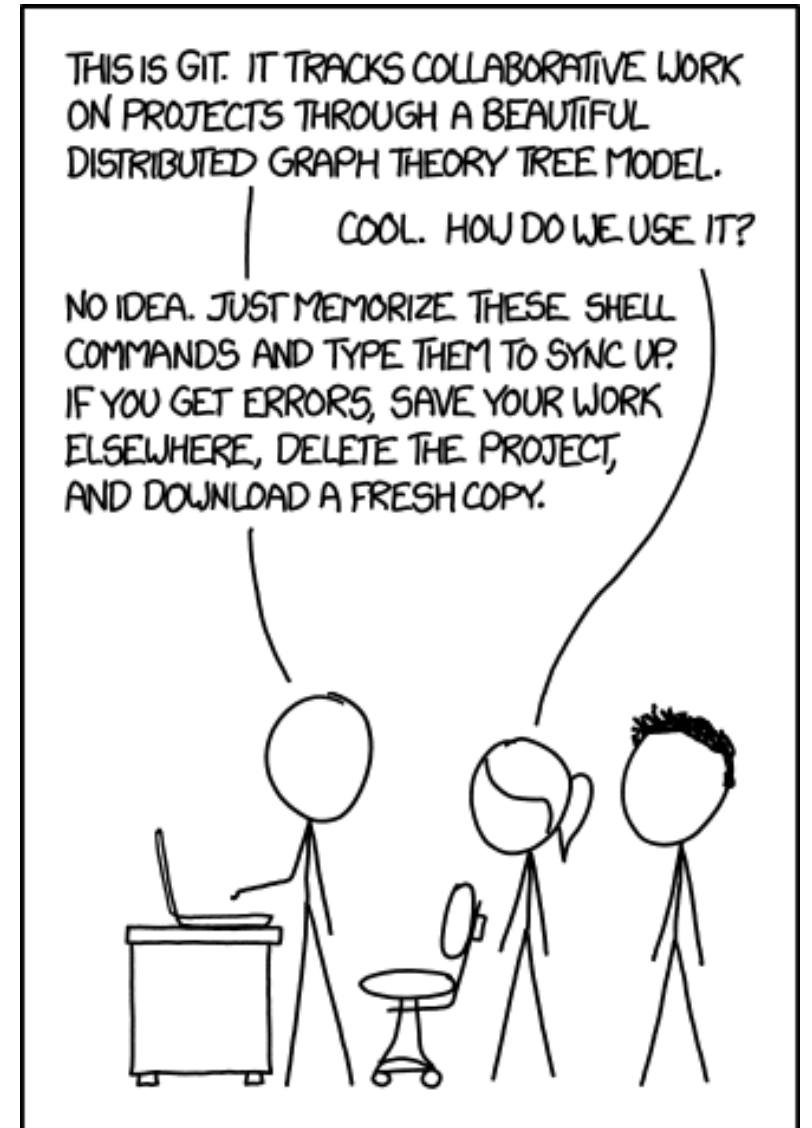
`git status`

`git add <filename> | -A`

`git commit -m`

`git pull`

`git push`



.gitignore

You don't want to store nonsense Eclipse files in your code repository!

Git lets you specify files that should be **ignored**.

- All you have to do is place a .gitignore file in the root of your repository.

A collection of ready-made .gitignore files is here:

github.com/github/gitignore

Some IDEs (Visual studio, IntelliJ,...) create a .gitignore for you.

Commit messages

Important because they create history

- Short and concise
- In **active voice**
 - “Do this thing”
- If applied, this commit will...
 - “Fix typo in introduction to user guide”

Great blog post: chris.beams.io/posts/git-commit/



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

GitHub Desktop

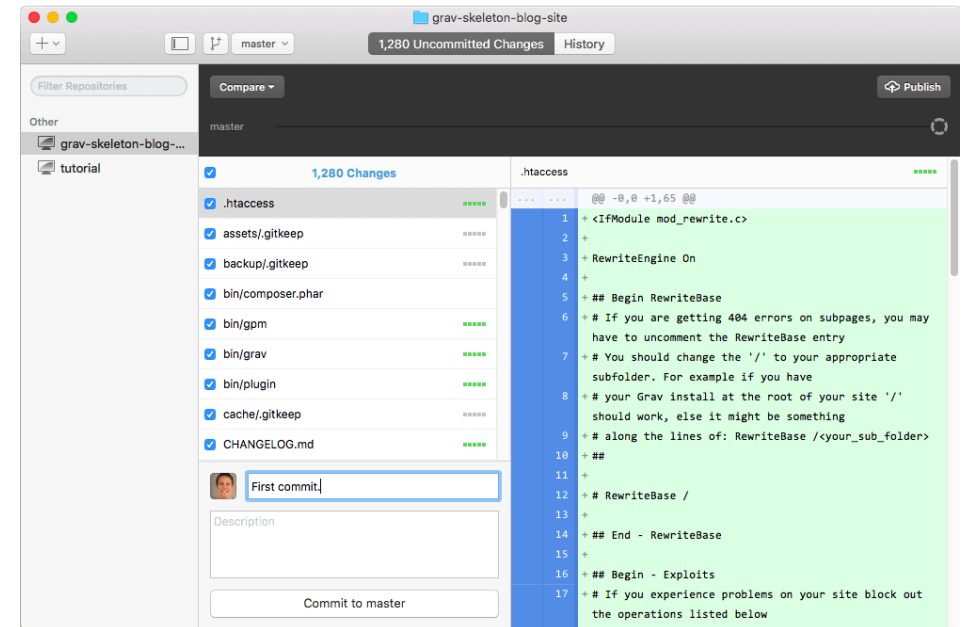
If you **hate** the command line, there are many applications for Git.

GitHub Desktop is the easiest to use:

- On Windows and Mac
- Comes in two versions
 - Old version is really good
 - New version is in beta
- Links with GitHub account
- Works with non GitHub repositories too

<https://desktop.github.com/>

GitHub Desktop



Git Resources

Git command line tutorial – Best starting place.

<https://try.github.io>

Pro Git – Perfect (and free!) e-book about Git.

[Pro Git](#)

First-Aid Git - Great place to look for quick solutions.

<http://firstaidgit.io/#/>

Git fix Um - Great place to search for solutions.

sethrobertson.github.io/GitFixUm