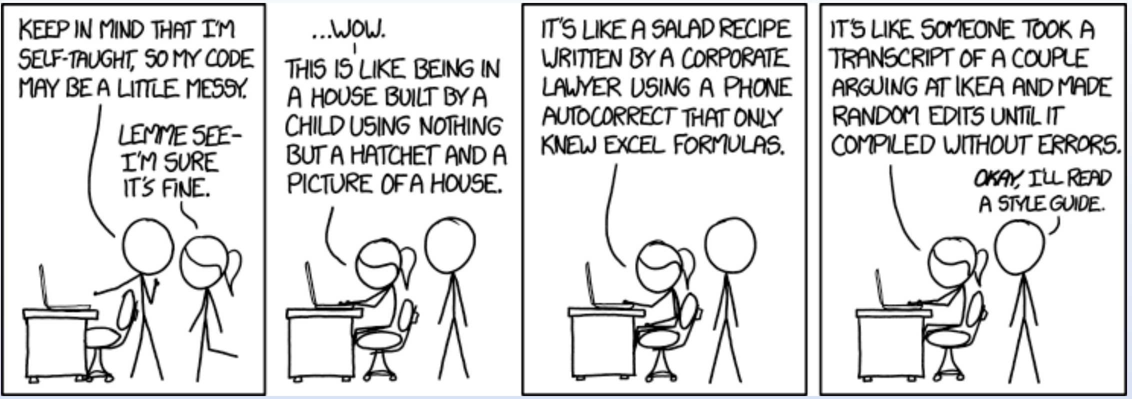
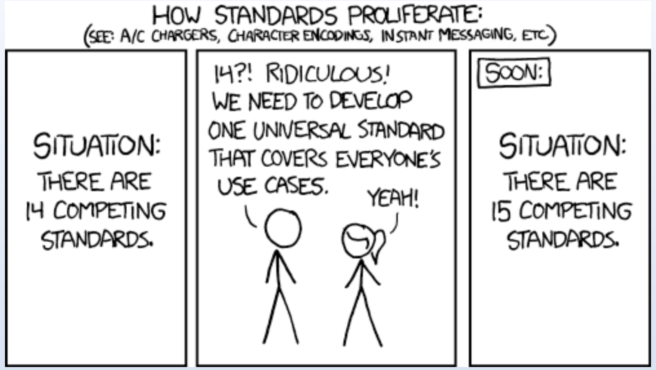
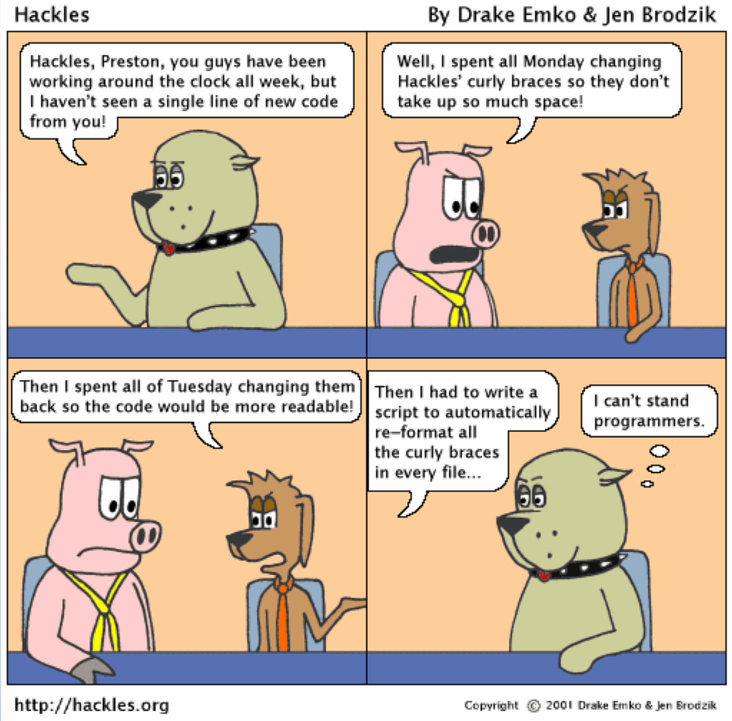

KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

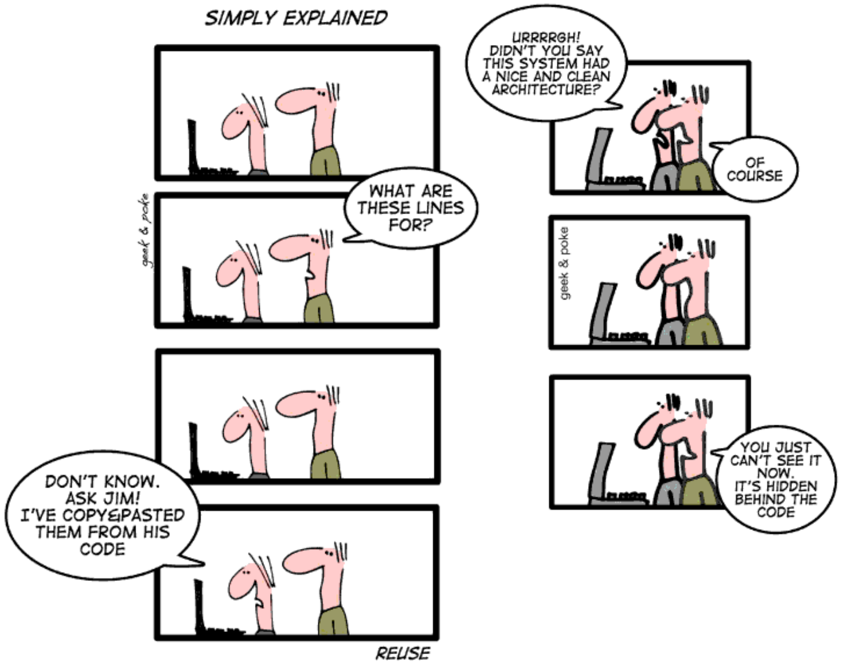

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.


Hackles
By Drake Emko & Jen Brodzik
Hackles, Preston, you guys have been working around the clock all week, but I haven't seen a single line of new code from you!
Well, I spent all Monday changing Hackles' curly braces so they don't take up so much space!
Then I spent all of Tuesday changing them back so the code would be more readable!
Then I had to write a script to automatically re–format all the curly braces in every file...
I can't stand programmers.
http://hackles.org
Copyright © 2001 Drake Emko & Jen Brodzik

Basic Coding Techniques

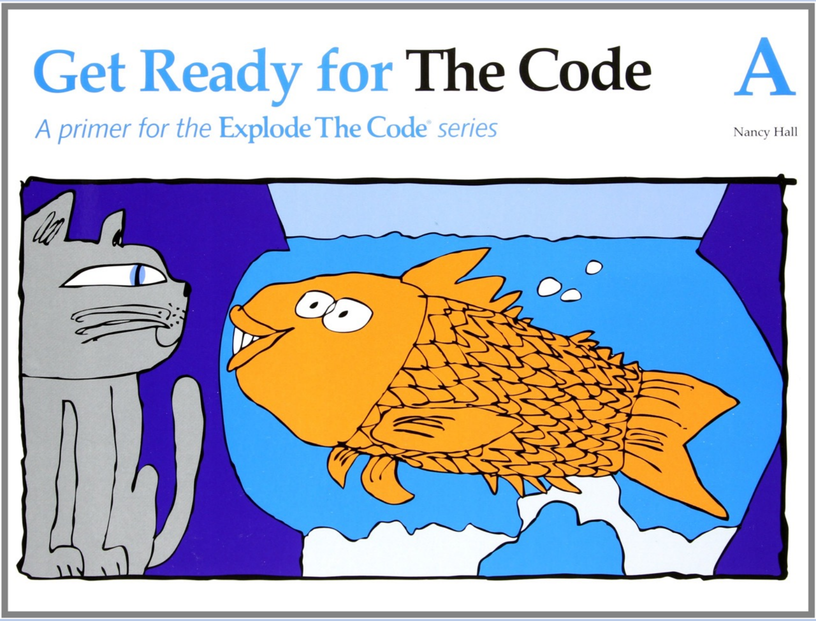
CSCI2340: (Graduate) Software Engineering

Steven P. Reiss


SIMPLY EXPLAINED
WHAT ARE THESE LINES FOR?
DON'T KNOW. ASK JIM! I'VE COPY&PASTED THEM FROM HIS CODE
URRRRGH! DIDN'T YOU SAY THIS SYSTEM HAD A NICE AND CLEAN ARCHITECTURE?
OF COURSE
YOU JUST CAN'T SEE IT NOW. IT'S HIDDEN BEHIND THE CODE
geek & poke
REUSE

Getting Ready To Code

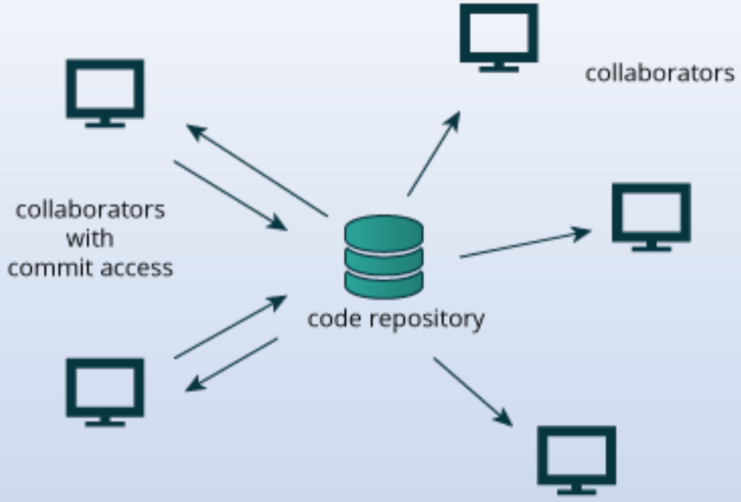
- We’re going to be doing design for a while
  - But will eventually start coding
  - And you should be coding the homework assignment
  - And you are starting to work on the project
    - If you need to do any prototyping
- This lecture is preparation for that



GIT

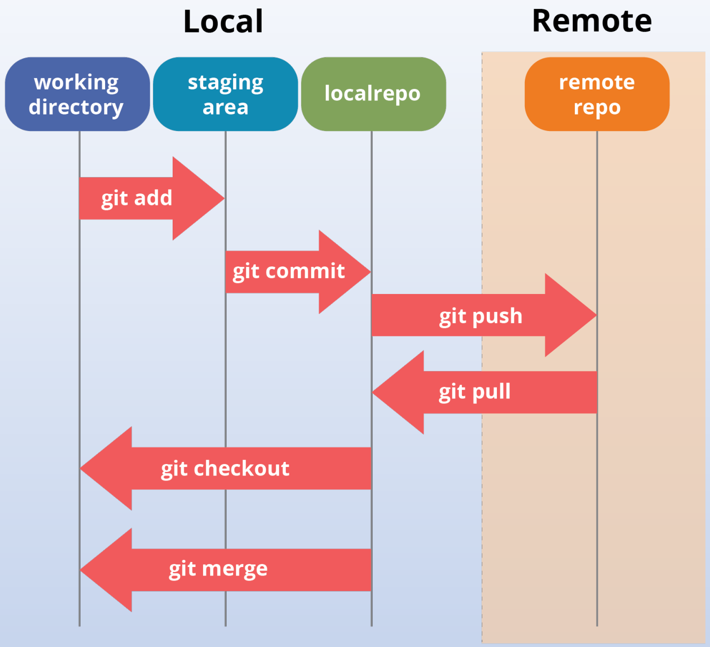
- We need support for joint projects
  - Source code control, version management
  - GIT is today’s standard (sccs, rcs, svn, perforce, …)
- GIT provides a flexible, adaptable platform
  - Distributed framework
    - Even for one person development
  - Allows lots of collaboration
  - Supports complex branch and merge operations
  - Provides the safety of older versions and going back
    - Use if even for one person development

GIT Basic Concepts



- Repository
  - Central location for all files
  - Can be GitHub, local, or anywhere accessible
  - GIT supports multiple repositories
    - Each user has their own repository
  - One is designated the master or head
    - Can be changed if needed
- Individuals clone (check out) the repository
  - They get their own copy of all the files
  - Comes with a link back to the cloned repository
  - But it is a repository in its own right

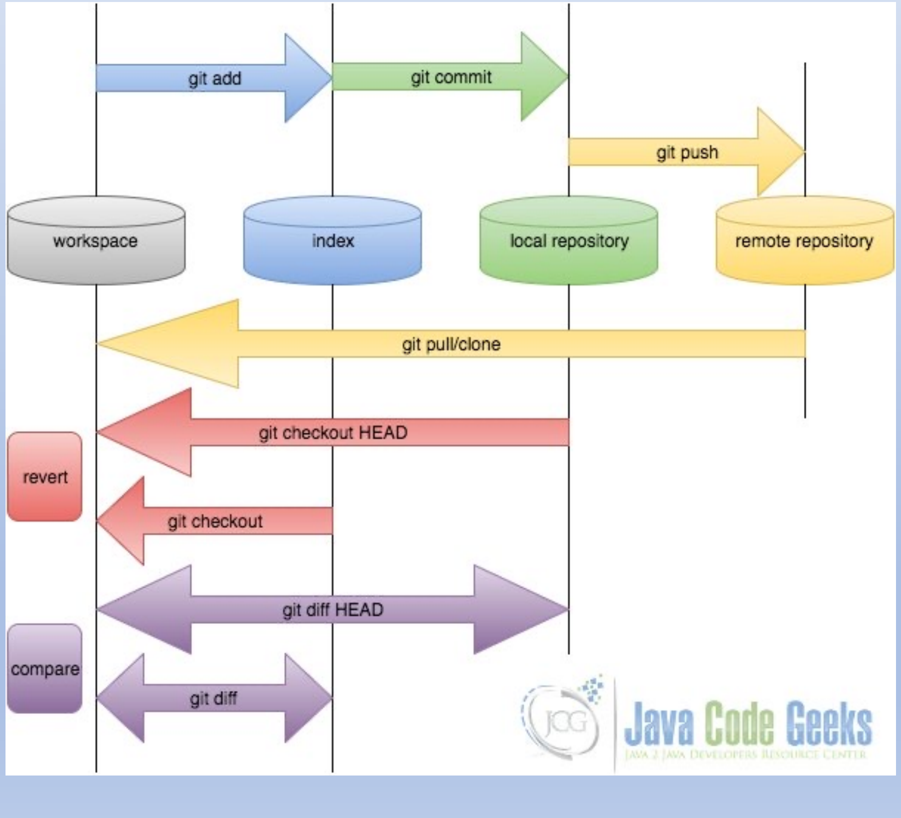
GIT Basic Concepts



- Individuals can edit their copies
    - Edit individual files
    - Create private files & directories
        - .gitignore file describes what is private
    - Create new public files & directories
        - But you must tell git using git add
    - Remove files & directories
        - But you should tell git using git rm
    - Status operation to check what has changed
    - Commit operation to commit changes locally
        - Puts the changes into the local repository
        - This allows going back, but doesn’t change the global repo

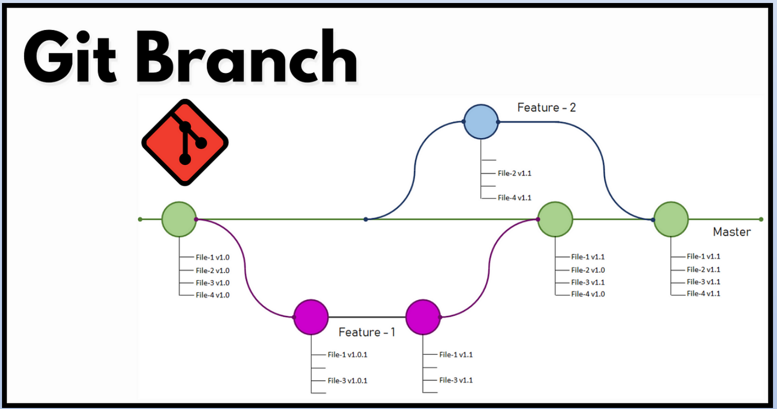
GIT Basic Concepts: Push and Pull

- Push a committed local repo to the global repo
  - Updates the global repo with changes
  - Creates new version of the global repo
  - Makes changes visible to others
- Pull global repo into local repo
  - Updates local files with global changes
- What happens if there are conflicts
  - Merge changes – doesn't always work
  - Can require manual intervention
  - Commit the merge code

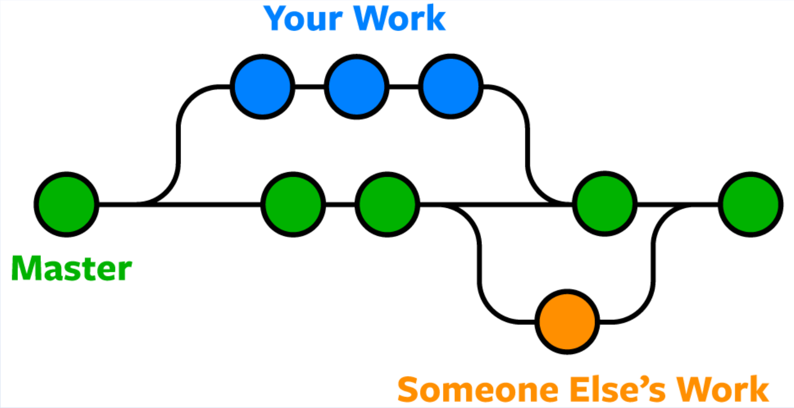


GIT Basic Concepts: Branches

- Sometimes you need to work independently
  - Don’t want all others to see your changes
  - Don’t want to see changes of others (temporarily)
  - But still want to save things globally
- Branches provide such a means
  - Branches are separate versions of the system
    - Independently developed from the main repository
    - But store in the repository
  - Branches can then be merged
    - With each other or with the main branch
  - We’ll cover these in more detail later
  - Today, *continuous integration* with a single branch is often used



GIT Complexities



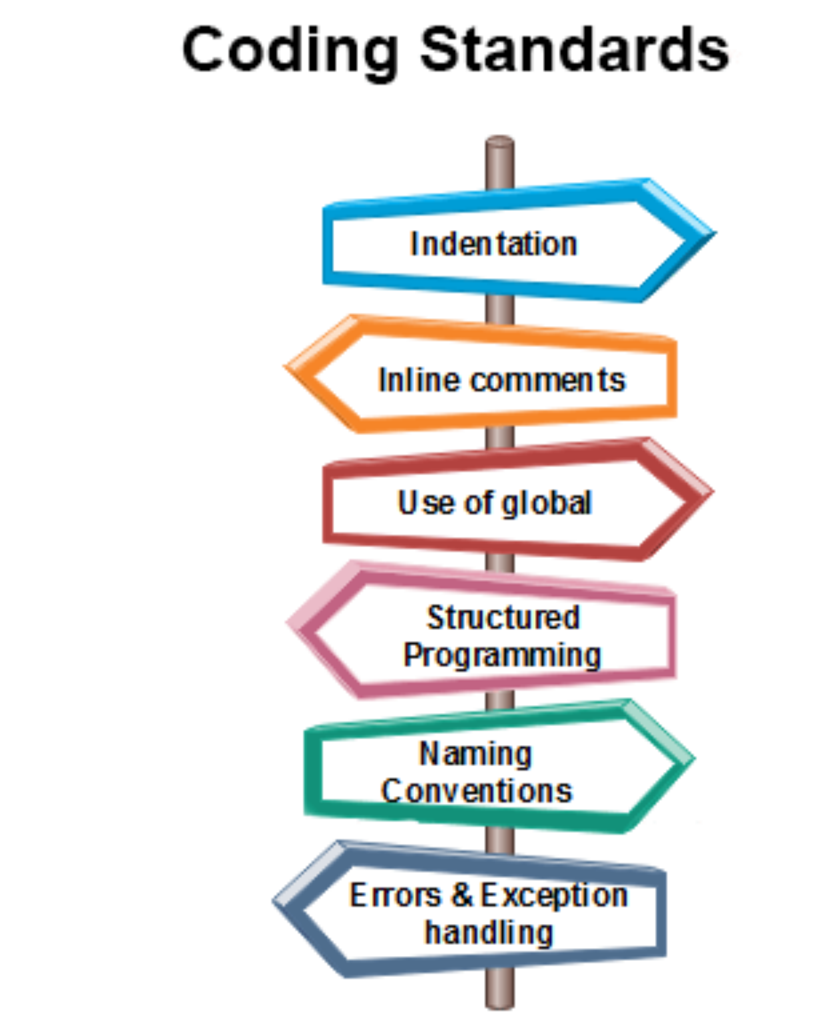
- Editing and merge conflicts
  - What if two people make changes to the same file
  - What if one person deletes a file someone else uses
  - You want to avoid this where possible – requires additional work
- This can happen with or without branching
- Repositories can be combined in various ways
  - Merge – merge the code from the two branches (safest; recommended)
    - But can require manual intervention and create corrupted files
  - Rebase – apply your changes to current version if possible
  - Stash and replace (removes all local changes) (git restore)
  - We'll get into these in more detail later
- You should read up on these and decide what to use
  - As a project team
- Time at end of class to set up GIT for your project

Coding Style

- Most frustrating part of collaboration & using open-source software
- Coding style is essential
  - Helps with maintenance
  - Helps with understanding
  - Helps to make the code readable to everyone
  - Open source should be open
  - Consistency in a large project
    - Much easier to work on code with known conventions
- WRITE CODE SO IT CAN BE READ BY PEOPLE
  - Not so it can be executed
  - For yourself, you team, and posterity
  - Be proud of your code

Coding Style Goals

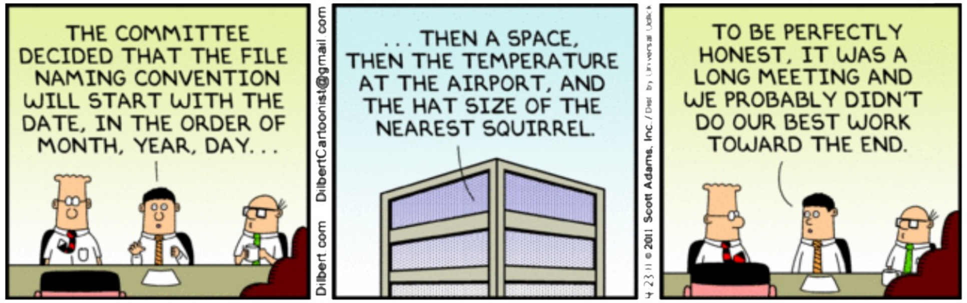
- Readability
  - Looks nice; easy to read; spaced appropriately
  - Easy to skim to get an overview of the code
  - Easy to find things (non-linear readability)
- Looking at an identifier
  - Easy to know what it is
  - Easy to know where it is defined
- Looking at code
  - One should understand its structure
  - Without having to read in detail
- Simplify debugging & maintenance
  - Avoiding name conflicts
  - Being able to find name in source
  - Make changes easier
  - Keep things local



Code Style Components

- Naming Conventions
- Ordering Conventions
- Coding Conventions
- Formatting Conventions
- File Organization

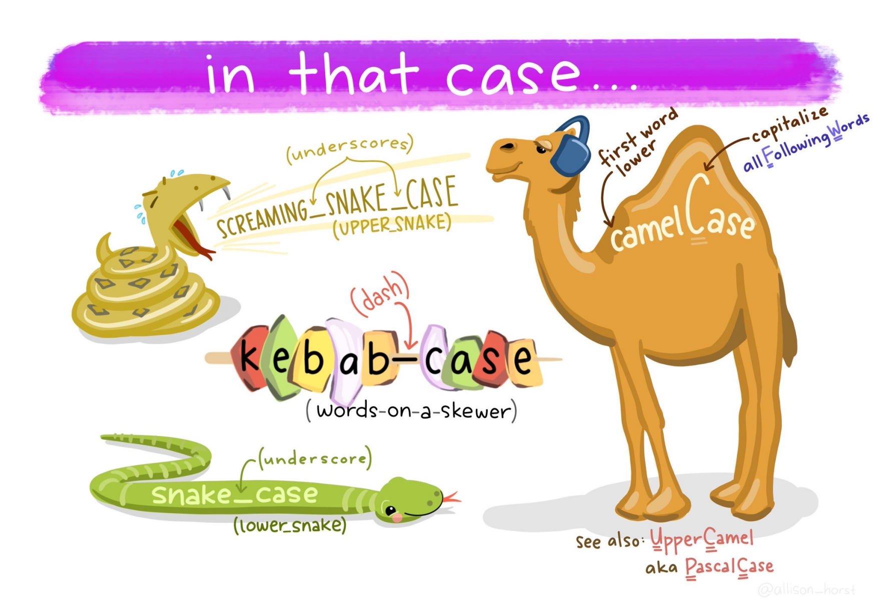
Naming Conventions


THE COMMITTEE DECIDED THAT THE FILE NAMING CONVENTION WILL START WITH THE DATE, IN THE ORDER OF MONTH, YEAR, DAY. . .
Dilbert.com DilbertCartoonist@gmail.com
. . . THEN A SPACE, THEN THE TEMPERATURE AT THE AIRPORT, AND THE HAT SIZE OF THE NEAREST SQUIRREL.
4-23-11 © 2011 Scott Adams, Inc./Dist. by Universal Uclick
TO BE PERFECTLY HONEST, IT WAS A LONG MEETING AND WE PROBABLY DIDN'T DO OUR BEST WORK TOWARD THE END.

Naming Conventions

- Goal:
  - Distinguish different types of names
  - Understand what the name means directly
  - Understand where to look for the definition
  - Understanding the scope of a name
- My coding style (not what you need to use)
  - Mainly for Java, adapted for other languages
- Fields (static variables)
  - All lowercase, contain an underscore, meaningful
  - Always private (or protected) to avoid naming conflicts
- Local Variables (and parameters)
  - All lowercase, no underscore
  - Short names okay if used within a few lines
  - Otherwise use meaningful names

Naming Conventions



- Constants (final static fields)
  - All uppercase, underscores separate words
  - Meaningful names
  - Start with package name if external
- Methods
  - Camel case names starting with lowercase
    - Can be single word (e.g., process), but this is unusual
  - Meaningful names
  - Access methods start with *get*, *set*, or *is*
  - Factory methods start with *create* or *new*

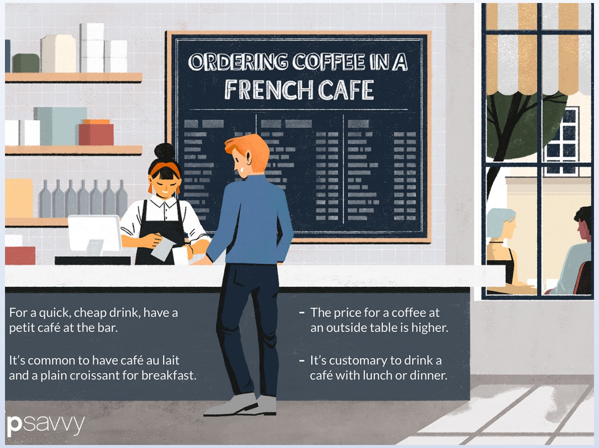
Naming Conventions

- Types (classes, interfaces, enums,…)
  - Camel case starting with uppercase (UpperCamel)
  - Outer types should start with package name
  - External (visible) inner types should start with package name
  - Single outer type per file
- Packages
  - edu.brown.cs.user.project.<package>
  - edu.brown.cs.project.<package>
  - Should always be there
- Imports
  - Use single class imports (not on-demand) [fix imports command]
  - Use static imports only where names will remain unambiguous

Naming Conventions

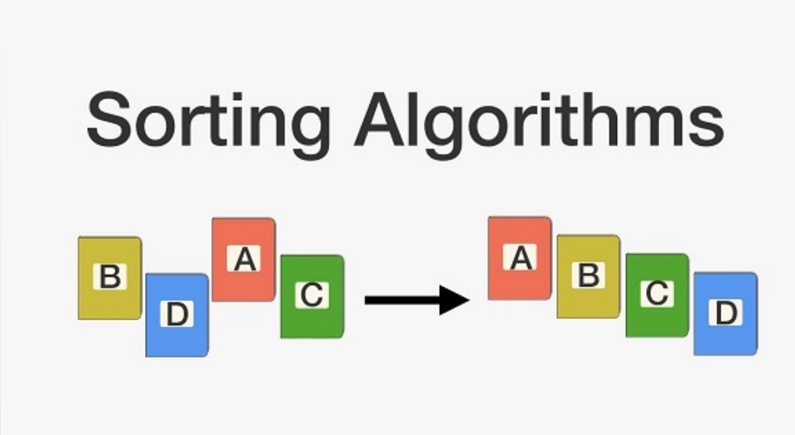
- Different language can require different conventions
  - May be recommended by the language
  - May be required by the language (dart _xx for private)
  - Adapt your coding conventions to the language as needed
  - Compromise on multi-lingual projects
- When modifying existing code, use its conventions
  - Augmenting, adding a feature
  - Patching, bug-fixing
  - You first should learn the existing code conventions
- When importing external code
  - Change to your conventions
  - But add citation to the original (copyright)
- Project should have a common set of naming conventions
  - Decide on these before you start coding (can use or adapt existing standards)
  - Write them down so all members of team are consistent
  - Put this in your repo

Ordering Conventions


ORDERING COFFEE IN A FRENCH CAFE
For a quick, cheap drink, have a petit café at the bar.
It's common to have café au lait and a plain croissant for breakfast.
- The price for a coffee at an outside table is higher.
- It's customary to drink a café with lunch or dinner.
psavvy

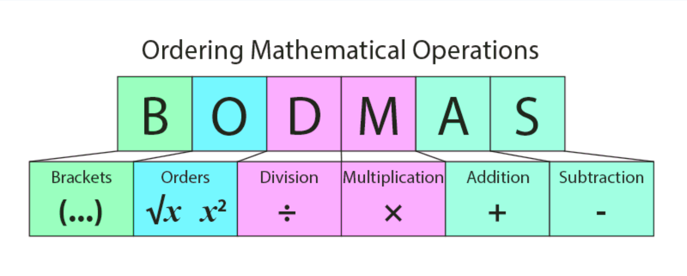
- Goal: Make it easy to find things in a file
- Within a file (my Java conventions as an example)
  - **Header comments** – name, purpose, author(s), copyright(s)
    - Should only have a single purpose (we'll get to that in class design)
    - Include names of all authors (add as needed)
    - Always include a copyright statement (even if simple or a reference)
  - Main method if present
  - Other top-level static factory methods
  - Field definitions (private and then static private)
  - Constructors and self-factory methods
  - Access methods
  - Public methods
  - Private methods
  - Inner classes (comment at end of class)
  - Tail comment -- note the end of the file

Ordering Conventions



- Not as strict as naming conventions
- Some slack allowed
  - Private methods related only to a public method
  - Inner classes related only to a single method
  - Factory methods for inner classes might be treated as constructors
  - Static methods might come earlier

Ordering Conventions



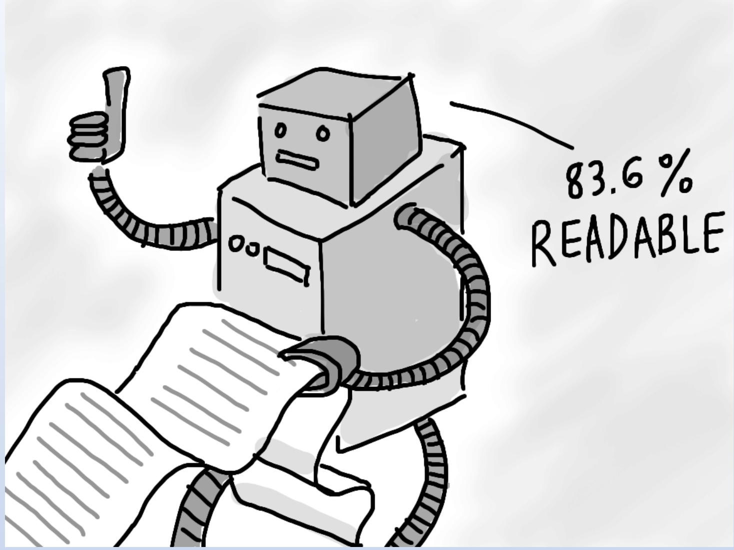
- Different languages can use different conventions
  - Some might recommend them
  - Some might require them (define before use)
- When modifying existing code, use its conventions
  - Augmenting, adding a feature
  - Patching, bug-fixing
  - You should learn the existing code conventions
- When adapting existing code, convert to your conventions
- Project should use a common set of ordering conventions
  - Decide on these before you start coding
  - Write them down & put into your repo

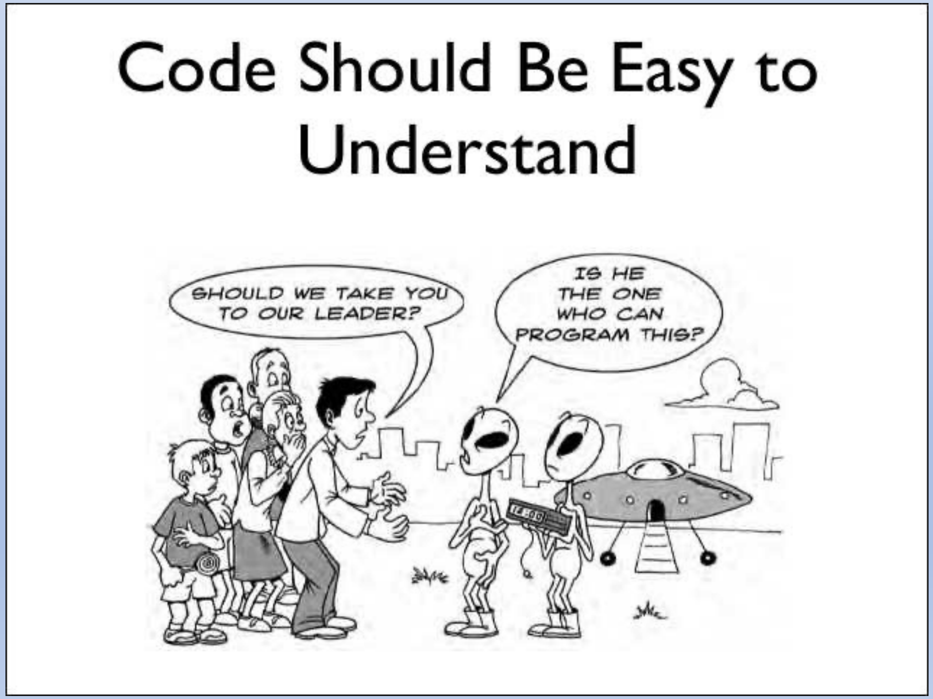
Coding Conventions

- Principle of least privilege (keep things local)
  - Make things private if possible
  - Make things public only if necessary
  - Fields should be private; protected at worst
    - Never accessed outside of a class (or subclass) directly
  - Javadoc (or equivalent) for all public and protected items
    - Non-trivial descriptions
  - Minimize public interfaces (keep small and few)
- Protected versus package-protected
- Methods should fit on one “page”
- Inner classes, static inner classes, and outer classes
- Hierarchies as inner classes versus outer classes
- Coding conventions depend a lot on the language
- We’ll cover coding conventions in more detail later

Formatting

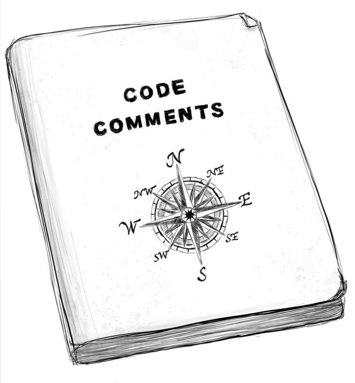
- Avoid complex conditions and constructs
  - Code should be obvious
  - Break up complex conditions into logical units
  - Parenthesize appropriately (X and Y or Z)
- Code should fit on the line
  - Maximum line length 80-100 characters
  - Split lines, or better yet, rewrite to avoid
- Use spaces, blank lines, and comments to enhance readability
  - And do so liberally
  - Sentences and paragraphs
- Consistent indentation
  - 3 or 4 spaces, not 2 or 8
- Consistent formatting (e.g., { ... })
  - And consistent spacing
- Consistent constructs (e.g., while(true) vs. for(; ;))




Code Should Be Easy to Understand

SHOULD WE TAKE YOU TO OUR LEADER?
IS HE THE ONE WHO CAN PROGRAM THIS?

Comments



- Block comments between logically separate components
  - Separate sections
  - Separate distinct functions
  - Make it easier to find the components
  - Make it easier to find things in the file without looking at all the code
  - I prefer enclosed comments that stand out
    - But some editors can make this hard (mine don't)
- Javadoc comments for external methods & fields
  - Or language-equivalent where appropriate
- In-line (//) comments where the code is non-obvious
- Use blank lines liberally but meaningfully (paragraphs, sentences)

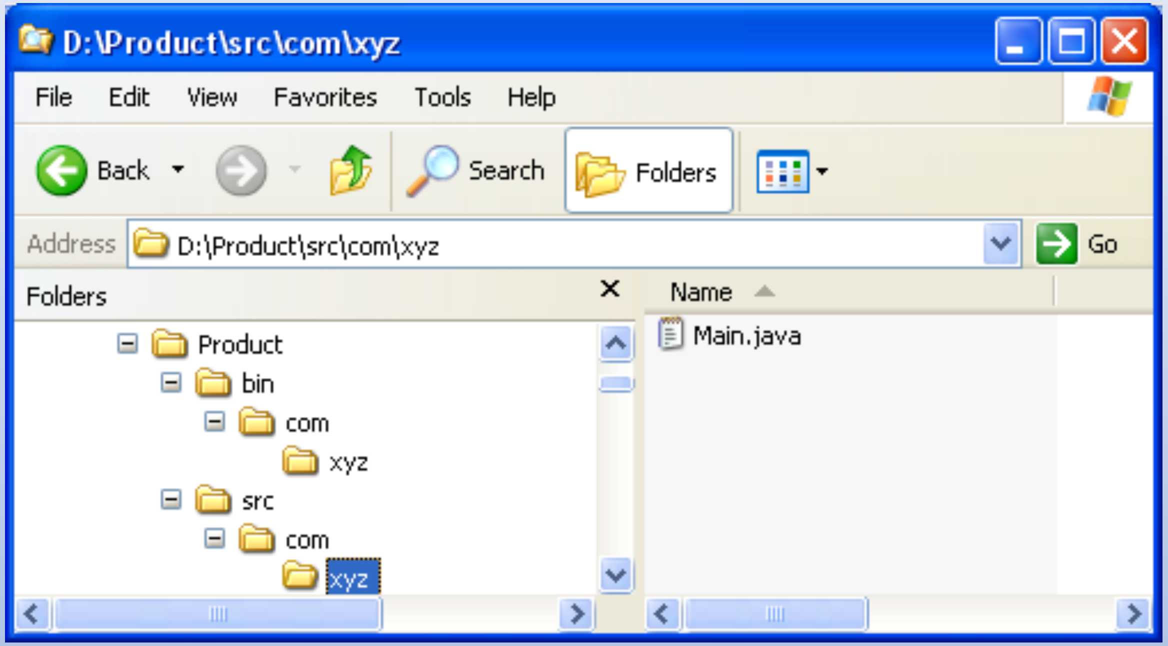
Formatting Conventions

checkstyle

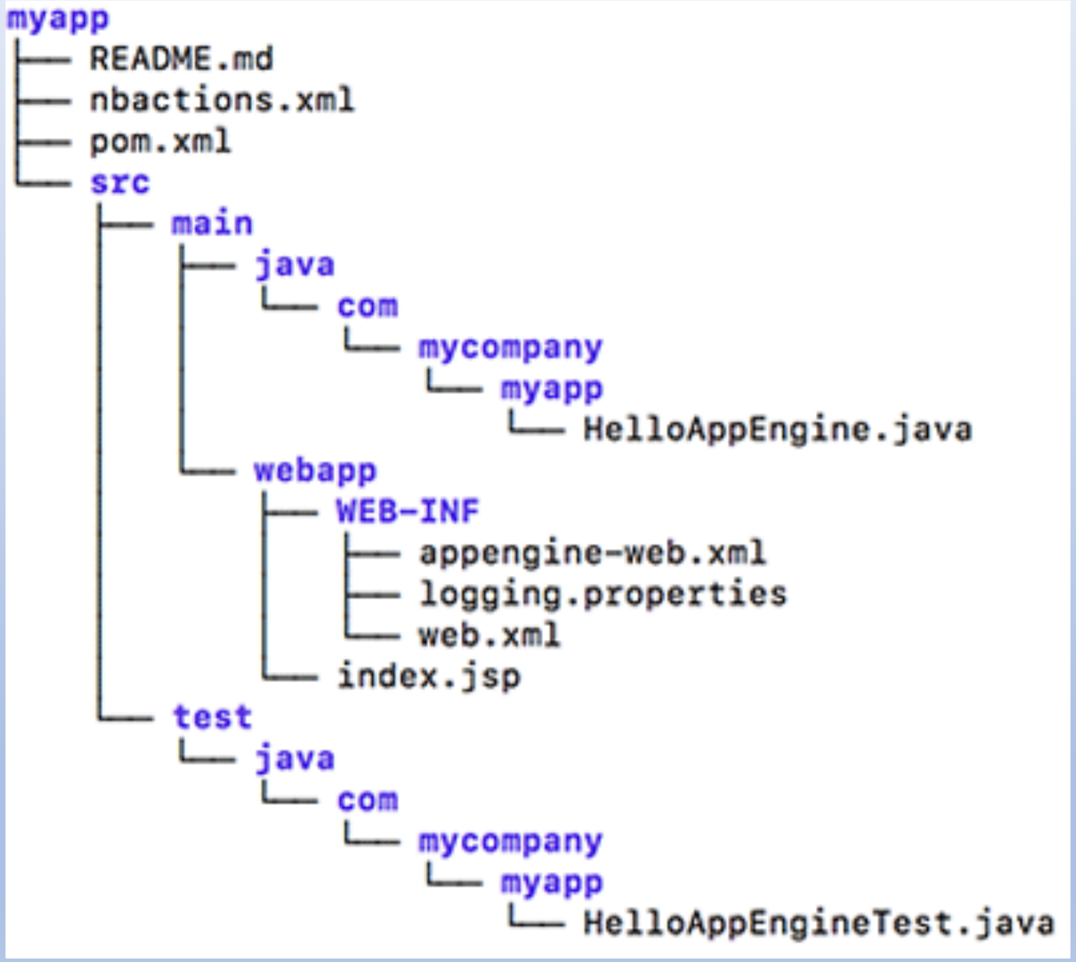
- Supported by IDEs
  - Eclipse, IntelliJ, various VS-Code plugins
  - Supported by checkstyle tool
- Environment can reformat code to set specifications
  - BUT not all conventions supported
  - Usually, will not change line spacing
  - Can handle initial indentation as you type
  - Can re-indent quickly
- You should set this up for your project
  - Write it down; include in repo
  - Define settings for the environments you are going to use

My File System Organization

- root (project name)
  - lib
  - resources
  - javasrc
    - edu …
  - java (compiler output)
  - bin
    - scripts and executables
  - <package>
    - src: link to ../javasrc/…/package
    - bin.java: link to ../java/…/package


D:\Product\src\com\xyz

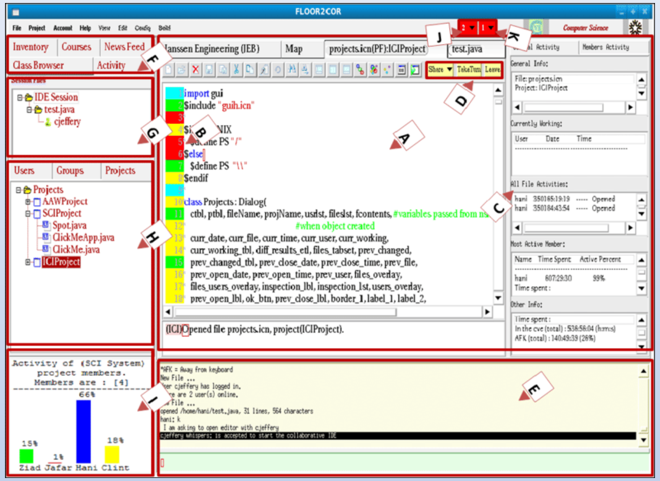
Maven File System Organization

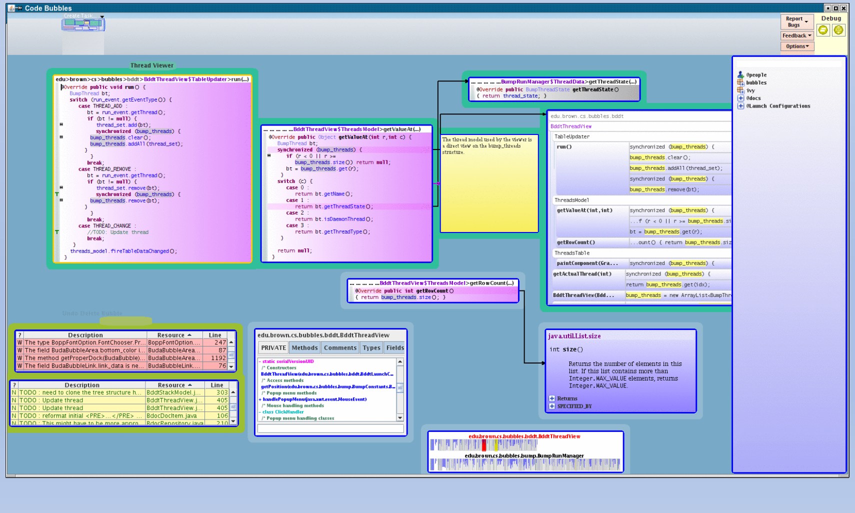


```
myapp
├── README.md
├── nbactions.xml
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   └── com
    │   │       └── mycompany
    │   │           └── myapp
    │   │               └── HelloAppEngine.java
    │   └── webapp
    │       ├── WEB-INF
    │       │   ├── appengine-web.xml
    │       │   ├── logging.properties
    │       │   └── web.xml
    │       └── index.jsp
    └── test
        └── java
            └── com
                └── mycompany
                    └── myapp
                        └── HelloAppEngineTest.java
```

Integrated Development Environments

- Help a lot, but take a bit of getting used to
  - Immediate feedback on syntax errors
    - Quick feedback on semantic errors
  - Good integrated debugging facilities
  - Tool integration (git, junit, ant, …)
  - Formatting, import organization
- I would like you to try Code Bubbles
  - If you are using Java
  - I need the feedback
  - Based on first assignment
  - But not required
- You should all be using IDEs





Research in Coding Techniques

- Automatic Style Inference and Application
- Evaluating readability based on style

PROJECT HOMEWORK

- Set up a GIT repo for your project
  - Instructions for project meeting to follow
- You should have a good sense of what to implement
  - Put a goal statement describing this in your repo
    - And hand-in via canvas
  - When you have a software architecture
    - Create a document describing it in your repo
    - This should be done by 9/26 if possible
- Agree on a coding standard for your project
  - Write it down (and save in your git repo)
  - Create an Eclipse / Idea / VSCode style file for it
  - Possibly create a CHECKSTYLE description for it

HOMEWORK /Further Reading

- Homework: Get an initial version of Bounce running
  - Use the coding style agreed upon for your project
    - Adapted for language differences
  - Convert to that style if necessary
  - Due 9/26 (hand in via canvas)
- Further reading
  - https://git-scm.com/docs
  - https://google.github.io/styleguide/javaguide.html
  - https://www.oracle.com/technetwork/java/codeconventions-150003.pdf
  - https://medium.com/@rhamedy/a-short-summary-of-java-coding-best-practices-31283d0167d3

Project Meeting Exercise

- You should have a GitHub account (prior homework)
- Meet as a project group
  - Decide on a project name (should have been done)
- One person create a GitHub repo for your project
  - Can be public or private
  - If private, add the other members of the group
    - Add me as well (github Id: stevenreiss)
  - Add a README.md file
  - Add a status directory
- Everyone in group clone the repository
  - Create a file tasks-<name>.md in status
  - Git add it to the repo
  - Git push
- When everyone is done
  - Git pull the repo so that everyone has the latest copy