Configuration Management

- Management of an evolving system in a **controlled** way.
  - **Version control** tracks component changes as they happen.
  - **System Building** assembles components for testing and release. Frequently is better.
  - **Change Management** addresses stakeholder proposals.
  - **Release Management** plans and prepares for distribution.
Version Management

- Two models
  - **Centralized** – master repository maintains all versions (SVN)
  - **Distributed** – multiple copies exist at the same time (Git)
- Features of both
  - Version and release identification
  - Change history recording
  - Independent Development
  - Project Support
  - Storage Management

Distributed Model Benefits

- A backup mechanism for the master repository.
- Allows developers to work offline
  - commit changes without a network connection
  - Developers can compile and test locally.
System Building

- Build script generation (configuration file)
- Build system integration with version control system
- Minimal recompilation (what changed or affected)
- Executable system creation
- Test automation (check build not broken by changes)
- Report success or failure of build and test
- Documentation (release notes) automatically generated

Change Management

- Ensure changes are applied in a controlled way.
  - requirements, bugs, …
- Consider factors in decisions to changes
  - Consequences
  - Benefits
  - Number of users affected
  - Cost
  - Product release cycle
Release Management

• Plan the release
• Prepare the system for release
  – Configuration files
  – Data files
  – Installation program
  – Electronic and paper documentation
  – Packaging and associated publicity
• Document the release

GitHub / Git

https://github.com
https://help.github.com/
GitHub/ Git

- **GitHub (server)**
  - On the internet, holds the master repo, issues, releases, ...
  - Code on master should always build/test/run with no problems
  - No changes are made directly in master
  - Proposed changes are reviewed, approved, then merged

- **Git (client)**
  - On a remote/local machine
  - Holds a clone of the master
  - Changes made in branches pushed back to the GitHub master and merged via a pull request

---

**Using GitHub and Git**

- **GitHub**
  - Select, estimate, and assign a pending issue issue.

- **Git**
  - Refresh (pull) or clone repo if needed. Create and checkout a new local branch in your repo clone.
  - Modify and add files and directories in the repo. Build and test before you commit.
  - Add your changes to the branch. Commit the branch with the #issue. Push branch to master.

- **GitHub**
  - Open a pull request for the commit for review by others.

- **GitHub**
  - Address any merge conflicts or comments. Merge the pull request and confirm.

---

https://guides.github.com/
Git Commands - local setup

# install git on your local system
# configure git username, email

git config --global user.name "[firstname lastname]"
git config --global user.email "[valid-email]"

# clone your team repo or the class repo

git clone [masterURL]

Git Commands - start a new branch

# update your local copy before you start

git pull origin master

# start a new branch

git branch [newbranchname]
git checkout [newbranchname] # never master!

# on a single line

git checkout -b [newbranchname]

# verify the branch just to be sure, never master

git branch
Git Commands - commit and push

# create, edit, rename, move, or delete files under “.”.
# build and test to verify changes work

```bash
git add . # add all changes to branch
git status # verify proposed changes are listed
git commit -m "closes #999" # associate with task 999
git push origin [branchname]
```

Git Commands - merge conflicts

# find files with merge conflicts on your local repo

```bash
git status
```

# edit files to resolve the conflicts between
# <<<<<<< HEAD and >>>>>>> BRANCH-NAME

# re build and test
```bash
git add .
```

```bash
git commit -m "resolved merge conflict"
git push origin [branchname]
```
GitHub Etiquette

- No changes made directly to master branch, never checkout master.
- All changes made in local/separate branches and merged via pull requests.
- All pull requests associated with an issue.
- Never break master. It should always build/test/run successfully.

Canvas - Brews

- README.md
  - Update team page in repo
- team/eID/README.md
  - Add your individual page to the repo
- Brews
  - Add a brewery to the tour
  - Use proper GitHub etiquette
  - When done correctly 1 addition and 0 deletions
  - May need to merge conflicts