Version Control

- A version control system (VCS) is a tool or system for keeping track of changes in files.
- A primitive form of VCS would be making a copy of a file every time you want to make a new version of the file. For example; lab1a.c, lab1b.c, lab1c.c, etc.
- The basic idea is to have some way to track multiple versions of a project including what the differences were, when they were made, and having a way to backtrack to those versions and undo changes if necessary.
- Many VCS have been developed over the years; *rcs*, *cvs*, *svn*, *git*. We will use *git*.

Version Control

- *git* is a different kind of VCS. *git* is a *Distributed VCS*. There is no centralized file server. Instead, all the contributors to a project have complete, separate copies of the project.
- Being decentralized also makes *git* a great tool to allow collaborators to all work on a project.
- Git is very popular and powerful. Its also well known for being hard to learn.
- There are many sources on the web to get help with *git*.
  - Software Carpentry
  - Github's "Hello World"
  - Git For Ages 4 And Up
  - You need source code control now

Version Control

- *git* isn't needed for this class much.
- Most will use it for *pulls* only.
- However, it is likely that if you do software, or hardware development work with hardware description languages, you will use a VCS.
- *git* can help you keep track of your work but don't use it to share your code with others.
- *git* is not a backup system, but used with the free remote repositories at `github.com` you can have a simple backup solution.

Version Control

- Getting setup to use *git*:
- First, installation:
- Debian-based distros:

```
    sudo apt-get install git
```

- Fedora:

```
    sudo yum install git
```

- Once you have it installed, check that its working:

```
git --version
```

- *git* should be found and should report the version

Version Control

- Setting up a repository:
- A repository is the place where *git* keeps track of your changes
  - All the files and directories you intend to keep track of.
  - The `.git` subdirectory holds all the tracked files recursively downward in the directory structure.
- The repository is stored alongside the files in the directory that you want to track.

Version Control

- Once you have it installed, check that its working:

```
git --version
```

- *git* should be found and should report the version
- Now, setup some necessary information for git:

```
git config --global user.name "User Name"
git config --global user.email "<user_name>@oregonstate.edu"
git config --global core.editor vim
```

- The git config commands will create a configuration file for you inside the `.git` subdirectory.
- Confirm this information is correctly in place:

```
git config --list
```

Version Control

- Once you are at this point, you can pull class code from github.

```
git clone https://github.com/rltraylor/sanity.git
```

- You have the entire `sanity` project code directory.
- It is a repository as the directory contains the `.git` subdirectory.

```
ls -a
```

- You can descend into the `sanity` directory and compile the C code there.

```
cd sanity
make
```

Version Control

- If you want to utilize more of what *git* can do for you, first create an account for yourself at at git-based repository provider such as: `github.com`.
- With `github.com`, as students, you can get unlimited private repositories for free (normally $7/month).
- Since our class work is to be individual work, I will expect you to use private repositories.

Version Control

- If you ever get messed up or simply want to forget having version control on a directory, simply remove the `.git` subdirectory.
- There is no other hidden infrastructure to find and/or remove.
- None of your files were ever touched by `git`.
- Let's take this action inside the `sanity` directory.

```
cd sanity
rm -Rf .git
```

- At this point, the `sanity` directory has no source code control.

Version Control

- Since we just removed the `.git` directory, it is just as if we had created the directory with `sanity.c` and `Makefile`.
- Let's say we did just create this directory and its contents and we want to place it under source code control.
- Within the `sanity` directory, we say:

```
git init
```

- This will allow *git* to trace changes to the directory and files in it.
- The data structures required for this are in the `.git` subdirectory.
- Notice the addition of (our own) `.git`

```
ls -a
```

Version Control

- When we compile, we make a lot of downstream files. We usually don't want to track these files, just our source code and makefile since the other files can be recreated.
- We can limit what files *git* puts in the repository by using a `.gitignore` file
- A `.gitignore` file for AVR development might look like :

```
*.o
*.map
*.bin
*.lst
*.hex
*.elf
*.srec
*.eeprom.hex
*.eeprom.bin
```

- *git* will not include or track these files for source code control.
- Create this `.gitignore` file

Version Control

- At this point, prior to adding anything to the repository, check the status of our directory:

```
git status
```

- git responds:

```
On branch master
Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        Makefile
        sanity.c
nothing added to commit but untracked files present (use "git add" to
```

- *git* is telling us that none of our files are tracked.

Version Control

- To begin tracking the files in our new repository, we type:

```
git add .
```

- Now all files at this directory and recursively below are being tracked.

```
git status
```

- git responds:

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   .gitignore
        new file:   Makefile
        new file:   sanity.c
```

Version Control

- Now the files are tracked but that they have not been committed to the repository.
- Put the added files/directories into the repository with the command:

```
git commit -m 'initial commit' (quotes not right as shown in .pdf)
```

- git responds:

```
[master (root-commit) 71fd74e] initial commit
 3 files changed, 100 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Makefile
 create mode 100644 sanity.c
```

- Now check the status again.

```
nothing to commit, working directory clean
```

- Our files and committed files are in sync

Version Control

- To track and store your work, first create a remote repository on github.com.
- Then associate your local repository with the remote one:

```
git remote add origin https://github.com/<user_name>/sanity.git
```

- git does the last step silently
- Now push your work to the remote repository with the command:

```
git push -u origin master
```

- You will be prompted for username and password
- Your work is now stored on the remote repository