Software Development on Linux Systems

4002-XXX-XX

By

Cody Van De Mark

Today

- Maintenance
- Patching
- Future Development

Maintenance

- Once you code is released, you will have to maintain it
- This includes:
  - Bug fixing
  - Feature requests
  - New developments
  - Security fixes
  - etc

Maintenance Tips

- Do not do development or patching on the master branch as you could ruin it and have nothing to fall back on;

  Instead create a new branch and develop on that

- You can merge it back it when it has been completely tested and is stable

Maintenance Tips

- Create a branch for each single purpose, such as fixing bug X
- It makes it much easier to see which branches have which fixes in them
- It makes it easier to merge code in when a bug is fixed, rather than waiting for each fix to become stable

Maintenance Tips

- Keep your development branch up to date

- This makes merging much easier and cleaner later

- This ensures your code does not become obsolete or invalid during development/testing

- Only commit back files you have changed;

  Class files, configuration files, notes and other files can be cause a problem if they are merged back in

Patching

- Patching code is when you create an individual fix for a problem
- Patches can be distributed and applied to existing code
- Normally, you merge a working patch back into your code if it is stable
- It is very common to also distribute an individual patch for users that already have your code so that they do not need to update

Patching

- Patching is actually fairly easy to do with working code
- A patch is just a diff between the changes you made to fix a problem and the code before the problem was changed
- If you have two directories, one before a problem was solved and one after, you can create a diff between them with

  ```
  diff  -crB  beforeFolder  afterFolder  >  changes1.patch
  ```
- This would create a file called **changes1.patch** that contains the differences

Patching

- You can also create a patch on a per file basis with:

```
diff  -cB beforeFile.py  afterFile.py  > changes2.patch
```

- This would create a file called **changes2.patch** that contains only the differences between those two files

- Actually applying the patch itself is just as simple

Patching

- To apply a patch, go into the directory you are patching and run the **patch** command

- This is done with:

```
patch  -p1  <  changes1.patch
```

- This would apply all of the changes from the patch to all of the code applicable in the directory

- The **p1** flag just is an indicator that the patch may not have been created on the same machine and should not be treated as if it was

Patching

- To remove a previous patch, you can use almost the same command that was used to apply it
- You need to patch that was used to apply the changes, to reverse them also
- This is done with the **-R** flag which means remove

```
patch  -p1  -R  <  changes1.patch
```

- This will remove all of the changes previously applied by that patch

  *Note: Another patch you applied could have one or two identical changes, thus removing partial changes of that patch as well

Future Development

- For future development, you should plan the changes that need to occur and prioritize them
- You may also find people post bugs and feature requests after you release your code
- You should also mark your plans in the bug tracker for group tracking purposes
- You are encouraged to continue development after this course is complete