R • I • T	Rochester Institute of Technology Golisano College of Computing and Information Sciences Department of Information Sciences and Technologies

4002-XXX
Software Development on Linux Systems
Lab 11- Setting up Bug Tracking, Patching, Forking and Merging

Name: _______________________________________ Section: _____________

In this lab you will be patching code and setting up bug tracking. When you create new projects, there are bound to bugs in the code and new features along the way. Often times bugs are reported by end users of the software. This is where bug tracking becomes important. With bug tracking you can communicate with end users, rate the severity of the problem and track the bug progress. Fixing bugs or tweaking features usually requires patching code. Code patches are created by determining the changes made and applying them to source code. End users then either download a new modified version of the code or the patch itself. In this lab you will create and apply patches, as well as create a bug tracker.

Activity – Patching

Patching is a fairly easy process. The idea of patching is to determine which files were modified from the original and what the modifications were. The new modifications are then applied to the old code.

1) Download **hellotest-1.4.tar.gz** and **hellotest-1.5.tar.gz** from MyCourses.

2) Uncompress these two directories. You should now have a folder called hellotest-1.4 and one called hellotest-1.5 in the same directory.

3) In the **hellotest-1.5** directory go into the **src** directory and open the files **hello.cpp** and **hello.h** .

   In the hello.h file replace the

   #define HELLO "Hello World"

   with

   #define HELLO "This is my updated code"

   and save the file

4) In the **hello.cpp** file add the line

   ```
   cout << HELLO << endl;
   ```

   several times after the first occurrence. Then save the file. The main function should now look like this:

   ```
   int main()
   {
           cout << HELLO << endl;
           cout << HELLO << endl;
           cout << HELLO << endl;
           cout << HELLO << endl;
           cout << HELLO << endl;
           return 0;
   }
   ```

5) Change directory to the one that contains both the hellotest-1.4 directory and the hellotest-1.5 directory.

   You will now create a patch of the newer code with

   ```
   diff  -crB  hellotest-1.4  hellotest-1.5  > ht1-5.patch
   ```

   This creates a new patch file containing all of the changes code between hellotest-1.4 and hellotest1-5. **It is important to note that any new files or folders are not recorded within the patch. The patch is only comparing files that exist in both folders. Any new files or folders must be committed to version control as normal.**

   With this patch, you are able to distribute the patch alone as a patch for end users or you can commit the updated code to version control to create an updated package.

6) Change directory to the **hellotest-1.4** directory

7) You will now apply the patch to the hellotest-1.4 folder with the patch command

   The hellotest-1.4 directory and the hellotest-1.5 directory that you downloaded were identical at the time you downloaded them. After we apply the patch to 1.4, the changes you made to the code in 1.5 will appear in 1.4.

   In the **hellotest-1.4** directory, run the following command

   ```
   patch  -p1   <  ../ht1-5.patch
   ```

   Note: That is the letter p and the number one in the patch command. This is a flag to ignore the fact that the patch may be being applied to a machine it was not created on.

8) Open the files **hello.cpp** and **hello.h** in the **src** directory of **hellotest-1.4**

   You should now see the changes you made to hellotest-1.5 in the files for hellotest-1.4

   You have successfully created and applied a patch.

   Instructor/TA Sign-Off ________________________________

9) You can also remove a patch that was previously applied. In the **hellotest-1.4** directory remove the patch with the following command.

   ```
   patch  -p1   -R  <  ../ht1-5.patch
   ```

10) Open the files **hello.cpp** and **hello.h** in the **src** directory of **hellotest-1.4**

   You should now see that the changes you applied in the patch have been removed and the code is back to state it was in before the patch was applied.

   You have successfully removed the patch.

   Instructor/TA Sign-Off ________________________________

Overview – Configuring Bug Tracking

There are many ways to setup bug tracking and distribute source code. You can setup your own systems or use existing web based ones. There are a number of online code hosting repositories that host open source software for free. There are also a number of bug tracking systems online, as well. In many cases, the code host also provides a bug tracking system. Some of the more popular ones are Github, Google Code, Launchpad and Source Forge. All of these systems provide bug tracking.

In this lab, you will choose one of them and upload a test package to their website. Then you will create a bug tracker for it.

The two systems covered in this lab are Launchpad and Github. Launchpad uses bzr while Github uses git. Launchpad allows you to automatically create packages for Ubuntu and distribute them, as long as they have the debian directory for deb files in the upload. Though you can automatically create deb files Launchpad, you may upload files for RPMs to Launchpad. You just won't be able to automatically make a deb file unless you have built the debian directory in your code. Many projects do not do this and hold the files for building RPMS or other packages in Launchpad instead.

Github is very similar to Launchpad, but uses git for committing. Github will not do automatic builds and is not tied to Ubuntu, but it is one of the most popular code hosting repositories. Github provides most of the features that Launchpad does just as effectively. You may find you like the Github interface and work flow better.

For this lab choose the system you would like to use. You will either use Git or Bzr.

Activity – Configuring Bug Tracking

Choose either the Launchpad section or the Github section for this activity

Launchpad

Launchpad provides a testing site called staging. Everything that happens on staging is deleted every Saturday. This provides us a place to test packages and Launchpad without any permanent changes.

1) Go to https://help.launchpad.net/Legal and read the terms of service

2) Go to https://staging.launchpad.net and choose **log in/register**

3) At the sign in page, choose **Create a new account**

4) Create an account using your RIT email address

5) Retrieve the confirmation code from your RIT email and paste it into the confirmation code text field on the Launchpad staging site. Now your account is ready to use.

6) Click on your account in the top right, to enter your account details.

7) In the user information area it will show a field for **SSH keys**, click the **+** sign to add a new key.

8) On your Linux system, create an ssh key with

   ssh-keygen -t rsa

   or use an existing key on that system if you have one.

9) Open the **id_rsa.pub** file in **~/.ssh/**

   Note: Make sure this is the .pub key. Otherwise you will be sharing your private key.

10) On the launchpad ssh keys page, enter the full contents of your **id_rsa.pub** under the **Add an SSH Key** text area.

   Note: You must include the entire contents of the pub file or else it will not work.

11) Click **Import Public Key**

12) Go to https://staging.launchpad.net/

13) Choose **Register a project**

14) For the name enter hellotest followed by your ID numbers, followed by your id initials. This is just to create a unique name for the lab.

    Example: If your RIT username is abc1234 then your project name would be

    hellotest1234abc

15) Enter the same name for the url

16) For the title enter **Hello World Test Lab**

17) For the summary enter, **This is a hello world example that is testing how launchpad works on staging.**

18) When asked if this is a duplicate, scroll to the bottom and choose **No, this is a new project.**

19) Leave the description blank

20) Leave the homepage blank

21) For license choose **GNU GPL v2**

22) Skip the box labeled "I do not want to maintain this project"

23) Hit **Complete Registration**

24) On the right choose **Configure bug tracker**

25) At the configure bug tracker screen, choose Bugs are tracked, **In Launchpad**

26) Scroll down and for Bug Supervisor click **choose** and click **pick me**

27) For Security contact click **choose** and click **pick me**

28) Choose **Change** at the bottom to confirm

29) At the main project page on the right choose **configure support tracker**

30) For tracker location choose **Launchpad**

31) At the main project page on the right choose **Configure translations**

32) For type of service for translations application, choose **Not Applicable**

33) Scroll to the bottom and hit **change**

34) At the main project page on the right choose **Configure project branch**

35) Choose **Create a new, empty branch and link to this series**

36) For branch name type in **Main**

37) Click **Update**

38) At the top of the main page choose **Code**

39) Download **hellotestdeb.tar.gz** from mycourses.

40) Uncompress this file somewhere on your Linux machine. This will uncompress to two tar files and a hellotestdeb-1.5 folder.

41) Open terminal and type

    bzr whoami "*firstname lastname* <*lastname@rit.edu*>"

    where *firstname* is your firstname, *lastname* is your lastname and *lastname@rit.edu* is your email. Make sure you includes the quotes and < > .

    This tells bzr who you are committing as.

42) Now type

    bzr launchpad-login *launchpad-id*

    where *launchpad-id* is your launchpad username. This is usually the prefix to the email you provided. If you need to check, click your name in the top right of launchpad.

43) Cd to the directory containing **hellotestdeb_1.5.orig.tar.gz, hellotestdeb-1.5.tar.gz** and the **hellotestdeb-1.5** directory

44) Initial bzr with

```
bzr init
```

45) Add hellotestdeb_1.5.orig.tar.gz and the hellotestdeb-1.5 folder with

```
bzr  add  hellotestdeb_1.5.orig.tar.gz  hellotestdeb-1.5
```

46) Commit the code to your local branch with

```
bzr  commit  -m "This is the initial commit"
```

-m just means set message, which is required for commits

47) Push your branch to Launchpad with

```
bzr  push  --use-existing  lp://staging/hellotest1234abc
```

where *hellotest1234abc* is the name your gave your project

48) On your projects Code page on Launchpad, click the branch link

It should look something like this

lp://staging/*hellotest1234abc*

49) On that page choose **Browse the code**

You should now see your uploaded files.

50) Now go back to your project Launchpad page and choose the **Bugs** page

51) On the right side click "Report a bug"

52) In the summary section field write "**Code prints extra lines of output**"

53) Hit the **next** button

54) For further information write "**The code outputs hello world more than it should. It is spamming output.**"

55) On the assign to field hit **choose** and hit **pick me**

56) Hit **submit bug report**

57) On the new page, click the expand arrow next to where it says bug affects

    This will give you a drop down list of properties.

    Find the **importance** dropdown and choose **Low**

58) Select **Save Changes**

59) Click on **Bugs** at the top of your project Launchpad page and it will take you to a list of all current open bugs.

    You have successfully setup bug tracking and a code repository.

Instructor/TA Sign-Off ________________________________

Github

1) Go to http://help.github.com/terms-of-service/ and read the terms of service

2) Go to http://help.github.com/privacy-policy/ and read the privacy policy

3) Go to https://github.com/signup/free and create an account using your RIT email

4) At the main Github page, click the **X** on the right to hide the tutorial windows

5) On the right you will see a section called **Your Repositories**. Click **New Repository**

6) For the project name enter hellotest followed by your ID numbers, followed by your id initials. This is just to create a unique name for the lab.

   Example: If your RIT username is abc1234 then your project name would be hellotest1234abc

7) Click **Create Repository**

8) Download **hellotestdeb.tar.gz** from mycourses.

9) Uncompress this file somewhere on your Linux machine. This will uncompress to two tar files and a hellotestdeb-1.5 folder.

10) Open terminal and type the following

    git config --global user.name "*Firstname Lastname*"

    where *firstname lastname* are your firstname and lastname

11) Then set your email with

    git config --global user.email ***your@email.com***

    where *your@email.com* is your RIT email

12) Click on your account in the top right of Github

13) At your profile page, choose **Edit Your Profile**

14) On the left choose the panel for **SSH Public Keys**

15) On your Linux system, create an ssh key with

    ssh-keygen -t rsa

    or use an existing key on that system if you have one.

16) Open the **id_rsa.pub** file in **~/.ssh/**

    Note: Make sure this is the .pub key. Otherwise you will be sharing your private key.

17) On the Github SSH Keys page choose **Add another public key**

18) For the **title** enter **Lab Key**

19) In the Key text area, enter the full contents of your id_rsa.pub.

    Note: You must include the entire contents of the pub file or else it will not work.

20) Hit **Add Key**

21) In terminal create a directory with the same name as the project

    mkdir *hellotest1234abc*

    where *hellotest1234abc* is your project name

22) Move the **hellotestdeb-1.5 folder** and the two tar files **inside** of your new folder

23) cd to the new directory. You should be inside of the *hellotest1234abc* folder where *hellotest1234abc* is your project name

24) Initialize Git with

    git init

25) Add the hellotestdeb-1.5 folder and the hellotestdeb_1.5.orig.tar.gz file to git

    git add hellotestdeb-1.5 hellotestdeb_1.5.orig.tar.gz

26) Commit the new files to your Git branch with a commit message

    git commit -m 'This is the initial commit'

    -m just means set message, which is required for commits

27) Add your Github repository location with

    git remote add origin git@github.com:*username*/*hellotest1234abc*.git

    where *username* is your Github username and *hellotest1234abc* is the name of your project.

28) Push you branch to your Github account with

    git push -u origin master

29) Now go to your Github repository and choose the **Code** tab

    You should now see the files you made have been committed to the repository

30) Go to the **Issues** tab

31) Choose **Create a New Issue**

32) In the **title** for the issue enter **"Code prints extra lines of ouput"**

33) Where it says **No one is assigned**, choose the dropdown and click your username

34) In the write text area, enter **"The code outputs hello world more than it should. It is spamming output"**

35) Select **Submit new issue**

36) Now click the **issues** tab of your repository. This is a list of current issues. The only one shown will be your newly submitted issue.

    You have successfully setup bug tracking and a code repository.

Instructor/TA Sign-Off ________________________________

Activity – Forking/Merging

In this activity, you will learn how to fork and merge projects you have created. This is often done is open source for development and testing. For example, if you would like to see how your code works with a different physics engine you make fork it, add the engine and test it. If it works, you would then merge it back it.

In fact, many projects are forked to make completely new projects based off of the previous code. These may merge in the future or may continue as a completely separate project possibly designed for a different purpose.

Choose either the Launchpad section or the Github section for this activity.

Launchpad

1) Find a class mate's project repository

2) On the main project page on the right click **Submit Code**

3) For the name of the branch enter **main**

4) Click **Register Branch**

5) You should now see the page of your personal branch of this project. At the bottom of the page under **Branch Information**, there will be a link to your class mate's project. It should be a link called hellotest*1234abc* where 1234abc is **your class mate's user id.**

   Note: You may want to open this in a tab, as you will be coming back to this page

6) On their project page, click **Code** at the top

7) Choose the **main** branch. It will be the one that **does NOT** have their user id in the path. It should also be the one that lists **Series: trunk**

   At the top of the page you should a line saying **Get this branch:**
   that has a branch command and address.

8) Create a folder on your machine called **clonetest**

9) Change directory to the clone test directory

10) Initialize bzr with **bzr init**

11) Type in the branch command shown on their project branch
It should say

```
bzr  branch  lp://staging/hellotest1234abc
```

where *hellotest1234abc* is the name of **their** project.

12) Open your newly downloaded code

Go into the **hellotestdeb-1.5** folder, then into the **src** folder

Open the file **hello.cpp**

13) Add the line

```
cout << "username" << endl;
```

after the first cout statement where *username* is your username. Then save the file.
The main function should now look like this:

```
int main()
{
        cout << HELLO << endl;
        cout << "username" << endl;
        return 0;
}
```

14) Now cd into ***hellotest1234abc*** folder where *hellotest1234abc* is the name of your **class mate's project.** This will be the folder that contains the **src** directory.

add the contents of the folder with

```
bzr  add  *
```

15) Commit your changes to your local branch with

```
bzr  commit  -m  'modified hello.cpp for more output'
```

16) Now we will go back to your branch of this project. You may have kept this open in a tab, but if not click on your account in the top right of Launchpad

17) Click the tab for **Code** at the top of your user information page

18) Click the main branch of the project you just forked. It will be the one called hellotest*1234abc* where 1234abc is **your class mate's user id**. It should be the one named main.

19) At the top of your forked branch page, it should list a command on how to push to your branch. Type that command in. It should say something like this:

    bzr push --use-existing lp://staging/~username/hellotest1234abc/main

    where username is your Launchpad user id and hellotest1234abc is the name of your class mate's project. Main is the branch name you were expected to give your forked branch earlier.

    This will commit all of the code back to your fork

20) On the Launchpad page for your fork, choose **Code** from the top

21) Click **Browse the Code** and you should see your updated code

22) Now make sure you are in the ***hellotest1234abc*** folder where *hellotest1234abc* is the name of your **class mate's project.** This will be the folder that contains the **src** directory.

    Merge with any updates from your class mate's original repository. His repository is referred to as **upstream** as it is where your fork came from. Since you downloaded the code from him, your bzr repository already points upstream. As long as you are within the directory you can automatically merge with

    bzr merge

    This will pull any changes from the original repository (upstream) into yours.

23) Re-add your merged files so that any changes they made can be pushed back to your repository with the updates from both of you. As long as you are in the hellotest1234abc folder where hellotest1234abc is the name of your class mate's project type

    bzr add *

24) Commit the merged files to your local branch

```
bzr commit  -m  'Merged code from upstream with current branch'
```

25) Push your code to your fork with

bzr push lp:staging/~*username*/*hellotest1234abc*/main

where *username* is your Launchpad id and *hellotest1234abc* is your fork name.

26) Now on the main page for **your** fork, choose **Code** at the top

27) On the code page, you should see an area titled **Branch Merges** under the Browse the Code link. In the **Branch Merges** section click the for **Propose for merging**

28) In the description area type "**Modified code for more output"**

Click the **Propose Merge** button at the bottom

29) Your collaborator now needs to go to the **Code** section of their main project page and choose **their** branch

In the section for **Branch Merges**, they should see a link that says
1 branch proposed for merging into this one

They need to click that.

You NEED to do the same for anyone who sent a merge request to you

30) They then have to click the link under **Branch Merge Proposal**

You NEED to do the same for anyone who sent a merge request to you

31) There will be merge commands listed as **To merge this branch:**

 On their machine, they need to go into the main directory for **their** repository (**NOT the fork they are currently working on)** and issue the command. They need to be in the main folder; the one that contains the **src** directory. The command should be similar to this.

bzr merge lp://staging/~*yourUsername*/*hellotest1234abc*/main

where *yourUsername* is your user ID (not theirs) and *hellotest1234abc is **their project name.***

You NEED to do the same for anyone who sent a merge request to you

32) They need to add the files into their commit cue with

```
bzr  add  *
```

You NEED to do the same for anyone who sent a merge request to you

33) They need to commit the merged files with

```
bzr  commit  -m  'merged files with another branch'
```

You NEED to do the same for anyone who sent a merge request to you

34) Then they need to push the files with

bzr push lp:staging/~*username*/*hellotest1234abc*/main

where *username* is theirLaunchpad id and *hellotest1234abc* is their project name.

You NEED to do the same for anyone who sent a merge request to you

35) Now they need to go back to the merge information page on Launchpad. Right above where the merge command was listed, there is a label called **Status: Needs Review.** They need to click the edit pen next to it and choose **approved.**

You NEED to do the same for anyone who sent a merge request to you

36) Your branch has now been merged back in. Launchpad has removed your forked branch.

    You no longer have a fork to continue working on, but in our case we will assume we want to continue working. To continue creating forks, you just repeat the exercise.

Instructor/TA Sign-Off ______________________________

37) **When both of you have completed the lab and received sign offs, you may delete your hellotest repository. Go to the main project page for <u>your</u> hellotest repository and hit the <u>Code</u> button at the top.**

    Click on <u>your</u> hellotest main branch when it comes up

    On the right click <u>Delete Branch</u>

    Then choose <u>Delete</u> at the permanent deletion page

Github

1) Click the **admin** button on the page for your project

2) On the left choose the panel for **Collaborators**

3) Add the username of one of your classmates and hit **add**

   Note: They need to do the same. This is just so both of you can commit back to merge your changes in.

4) Find a class mate's project repository

5) At the top of the page hit **fork**

6) Create a folder on your machine called **clonetest**

7) Change directory to the clone test directory

8) Initialize git with **git init**

9) Clone your repository onto your machine with

   git clone git@github.com:*username/hellotest1234abc*.git

   where *username* is **your** username and *hellotest1234abc* is the name of **their** repository, that you forked.

   You should now have the repository on your local machine

10) Open your newly downloaded code

    Go into the **hellotestdeb-1.5** folder, then into the **src** folder

    Open the file **hello.cpp**

11) add the line

    cout << *“username”* << endl;

    after the first cout statement where *username* is your username. Then save the file. The main function should now look like this:

```
int main()
{
        cout << HELLO << endl;
        cout << “username” << endl;
        return 0;
}
```

12) Now add your modified code to git with

```
git add hello.cpp
```

13) Commit your change with

```
git commit  -m 'modified hello.cpp for more output'
```

14) Push your changes back to your forked repository with

```
git push -u origin
```

15) Now you can also grab changes, merge and commit to the person you forked from if they have added you as a collaborator.

    The code you have forked from is known as **upstream**

    Add them as your upstream with

    git remote add upstream git://github.com/*theirUsername/hellotest1234abc.*git

    where *theirusername* is the username of the person you forked from

    and *hellotest1234abc* is the name of **their** project.

16) Now fetch the most current changes from **upstream** with

```
git  fetch  upstream
```

17) You want to merge these changes with your changes to ensure you do not overwrite any of their changes.

    To do this type

```
git  merge  upstream/master
```

18) Now go to the top level directory containing the **hellotestdeb-1.5** folder and the tar.gz file from your fork.

    Re-add your merged files so that any changes they made can be pushed back to your repository.

```
git add *
```

19) Commit your merged files so we can push

```
git commit  -m  'Merged code from upstream with current fork'
```

20) Push your code to your fork with

```
git push -u origin
```

21) On the main project page of your fork click the button for **Pull Request** at the top right

    Do NOT hit the tab called Pull Requests. That is for requests sent to you. The Pull Request button is above that to the right.

22) You will see a title field and a text area

    In the title field enter **"added new output"**

    In the text area enter **"This added a new line of output to the hello.cpp file in src"**

23) Click the button for **Send Pull Request**

24) Your collaborator should now see a one next to the tab for **Pull Requests**

They should click on the tab for **Pull Requests (1)** on **THEIR** project page

They should click on the name of the pull request shown

Here you can discuss the pull requests.

You NEED to do the same for anyone who sent a pull request to you

25) They should now add you to their remote branches with

git remote add *username* git://github.com/*username*/*hellotest1234abc*.git

where *username* is the username of whoever is requesting from them and *hellotest1234abc* is their project name. The first occurrence of this in the command is just the title you are giving the remote branch so you can reference it.

You NEED to do the same for anyone who sent a pull request to you

26) They should now fetch your branch with

git fetch *username*

where *username* is the username of whoever requested from them.

You NEED to do the same for anyone who sent a pull request to you

27) They should merge your changes with theirs with

git merge *username*/master

where *username* is the username of whoever requested from them.

If there are any conflicts open up the conflicting files and you will see the lines with conflict have been listed. Remove the conflicts and comments, do a git add. Then do a git commit. Then you can do the merge command again successfully.

You NEED to do the same for anyone who sent a pull request to you

28) Now they need to push back to the repository with merged code with

```
git push -u origin
```

You NEED to do the same for anyone who sent a pull request to you

29) Now if you go to **their** project page and look at the code, you will see that the code is merged with the code you sent them.

30) Now you should update your code with the upstream so that you also have all the merged code.

This is done the same way you did when you originally grabbed their updates

```
git  fetch  upstream
```

31) Now merge their newest updates with your code local branch

```
git  merge  upstream/master
```

32) Change directory to the top level directory containing **hellotest-1.5/** and the **hellotest_1.5.orig.tar.gz**

Add your new merged files to your local branch

```
git add *
```

33) Commit these merges to your local branch with

```
git commit  -m 'Merged with upstream'
```

34) Now push your merged files back to your forked repository with

```
git push -u origin
```

You have successfully forked a project, modified and merged your changes upstream. Now you have also updated your forked repository to match the upstream one.

Instructor/TA Sign-Off ________________________________

35) **When both of you have completed the lab and received sign offs, you may delete your hellotest repository. Go to the main project page for your hellotest repository and hit the <u>admin</u> button.**

Scroll to the bottom and find the option for <u>Delete this repository</u>

Click <u>Delete this repository</u>

Then choose <u>I understand, delete this repository</u>