

GIT - EGIT

Exercises

Exercise 1 – GitHub Setup



Overview

In this exercise, you will setup Github for Windows (or Mac) and configure Eclipse with GIT

Objectives

At the conclusion of this exercise, you should be able to:

- ✓ Install GitHub for Windows (or Mac)
- ✓ Explore Git in Eclipse
- ✓ Use the Git command line
- ✓ Create an initial repository
- ✓ Explores the repository in both Cmd line and Eclipse

Step 1: Preparation



In this exercise, you will obtain and install a GitHub environment. Begin by downloading the GitHub implementation for your platform.

Either Windows:

<https://windows.github.com/>

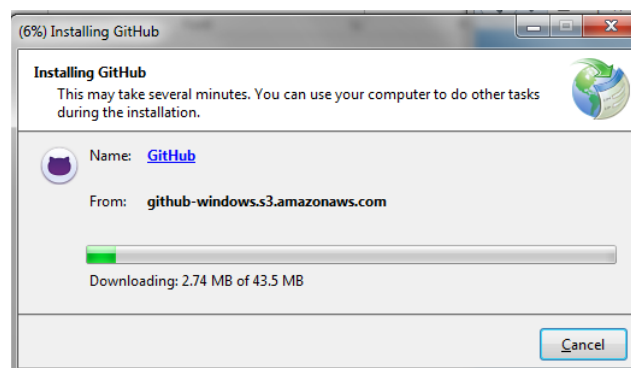
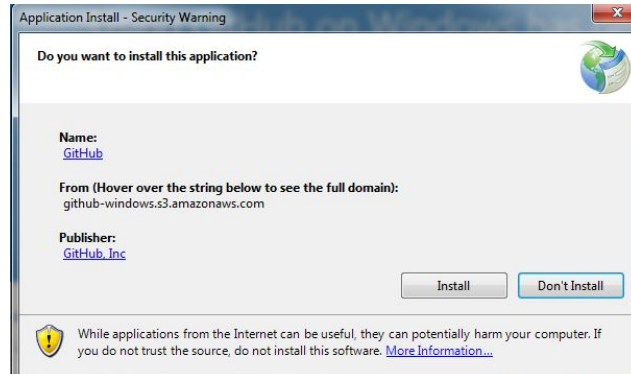
Or Mac:

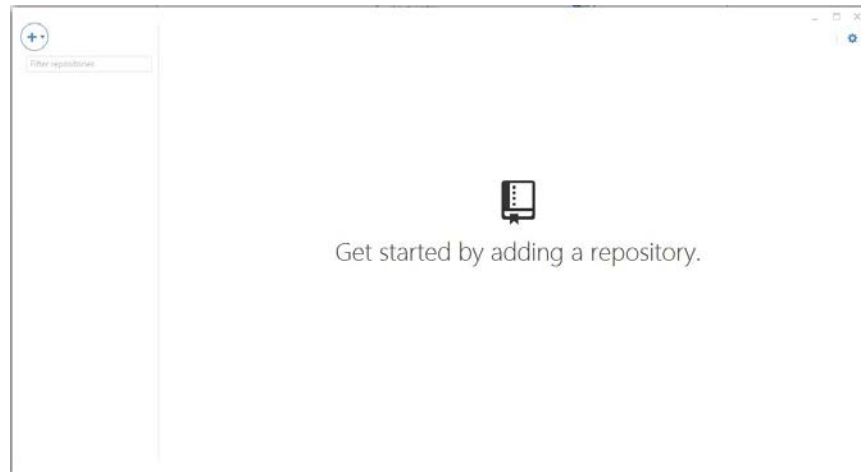
<https://mac.github.com/>



Step 2: Install GitHub

Run the installer, accepting the defaults. (Windows screens are shown, Mac will differ slightly)



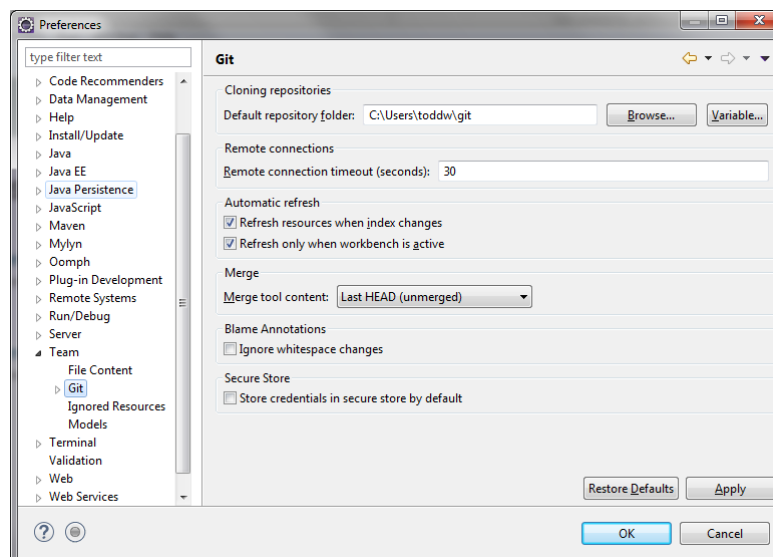


Close the GitHub for Windows (or Mac)

Step 3: Interact with Eclipse GIT

Open Your Eclipse editor in a new workspace.

Open the Preferences -> Team -> Git



The settings should be pre-set, but a few are worth mentioning:

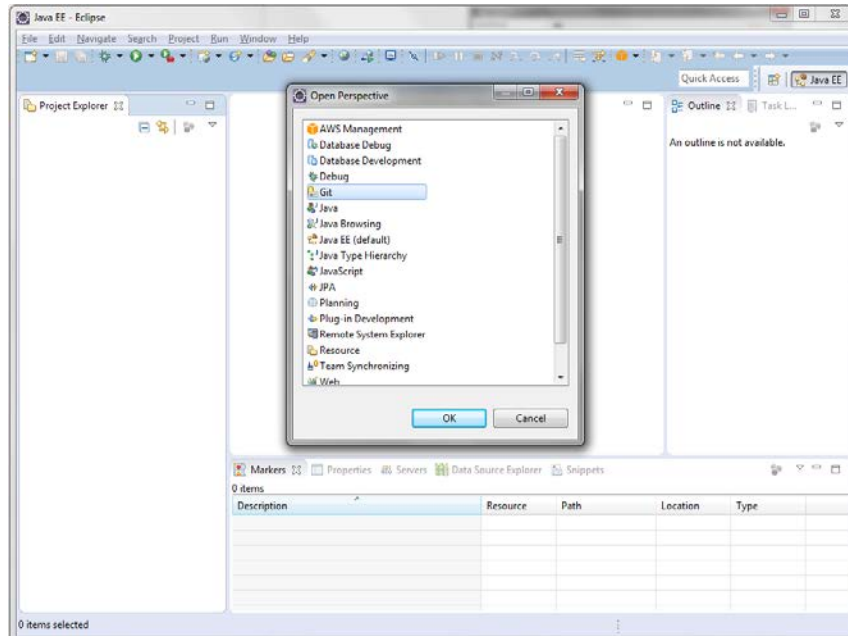
Default repository folder - Defines where the editor should prefer repositories to exist

Remote Connection Timeout - For slow servers you may need to increase this.

Cancel out of the preferences

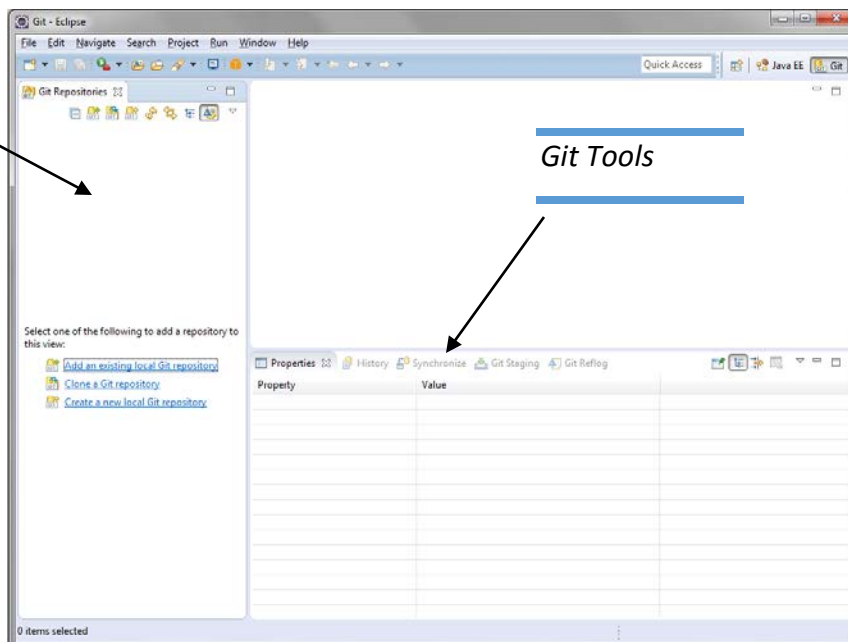
Step 4: GIT Perspectives

Return to the Eclipse workbench and open the Git perspective.



The Git perspective displays very useful tools:

*Display all
repositories*



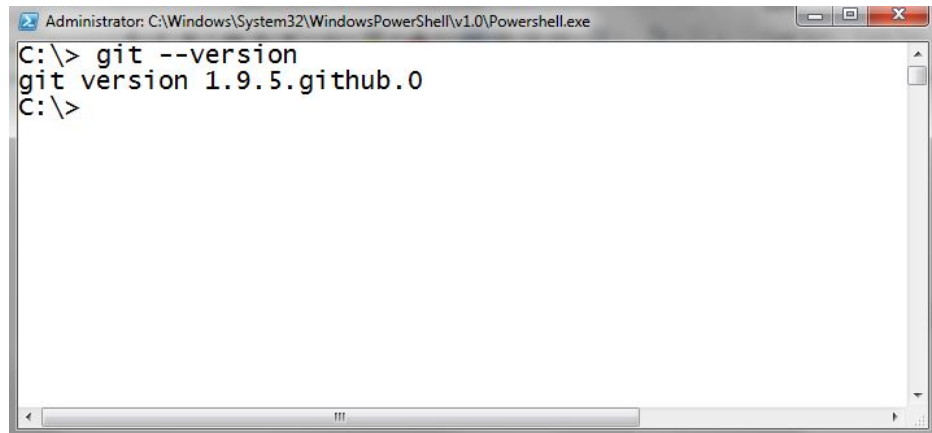
Step 5: Use Git Shell

From the Start menu (or applications) open a GIT shell.

- The GIT shell is a normal shell window with the environment set to include GIT.

Verify the version of Git with the following command

```
git --version
```

A screenshot of a Windows PowerShell window titled "Administrator: C:\Windows\System32\WindowsPowerShell\v1.0\Powershell.exe". The window shows a command prompt with the text "C:\> git --version" on the first line, "git version 1.9.5.github.0" on the second line, and "C:\>" on the third line. The window has a standard Windows title bar with minimize, maximize, and close buttons.

Step 6: Set common GIT Options

Git has many core options that change the behavior of Git. Display all the settings for Git:

```
git config --list
```

The first thing you should do when you install Git is to set your user name and e-mail address. This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating:

```
git config --global user.name "John Doe"  
git config --global user.email johndoe@empl.com
```

```
git config --list
```

Step 7: Create a local Git Repository

In your GIT shell change to c:\temp and run the following command to create a project folder:


```
mkdir c:\temp\gitclass
```

Change into that folder:

```
cd c:\temp\gitclass
```

Initialize the git repository for our project directory with the following command:

```
git init
```



```
posh-git - gitclass [master]
C:\>
C:\> cd \temp
C:\temp> mkdir gitclass

Directory: C:\temp

Mode                LastWriteTime         Length Name
----                -
d-----          3/9/2015   7:50 PM             gitclass

C:\temp> cd gitclass
C:\temp\gitclass> git init
Initialized empty Git repository in C:/temp/gitclass/.git/
C:\temp\gitclass> [master]>
```

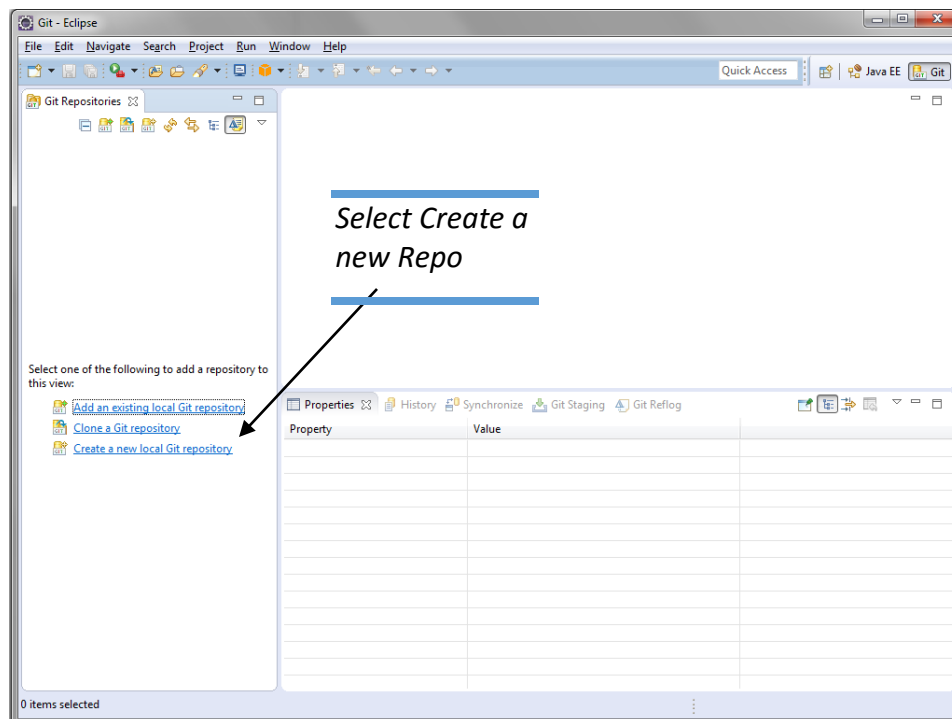
Compared to SVN, the `git init` command is an incredibly easy way to create new version-controlled projects. Git doesn't require

you to create a repository, import files, and check out a working copy. All you have to do is `cd` into your project folder and run `git init`, and you'll have a fully functional Git repository.

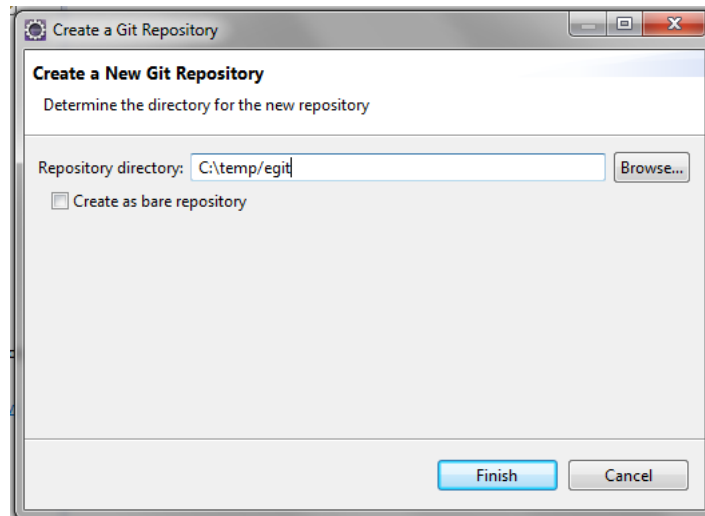
However, for most projects, `git init` only needs to be executed once to create a central repository—developers typically don’t use `git init` to create their local repositories. Instead, they’ll usually use `git clone` to copy an existing repository onto their local machine.

Step 8: Create a local Git Repository in Eclipse

In Eclipse open the Git perspective, and select the "Create a new Git Repository" link in the Git Repositories view.



In the popup dialog put c:\temp\egit in the Repository Directory box:



The new repository will be added to the view and you can explore the parts of the repository.

Exercise 2 – Managing Files in Git



Overview

In this Lab you will add and manage files in your "egit " git repository. We will then clone the repository to mimic the actions developers would normally take.

Objectives

At the conclusion of this exercise, you should be able to:

- ✓ Manage files within the git repository
- ✓ Make changes and commit to the repository
- ✓ Clone a repository for developer actions

Step 1: Add content to the main repository



In Eclipse create a new "General -> Project" named egitproject. Create a file named file1.txt with a simple text message inside it :

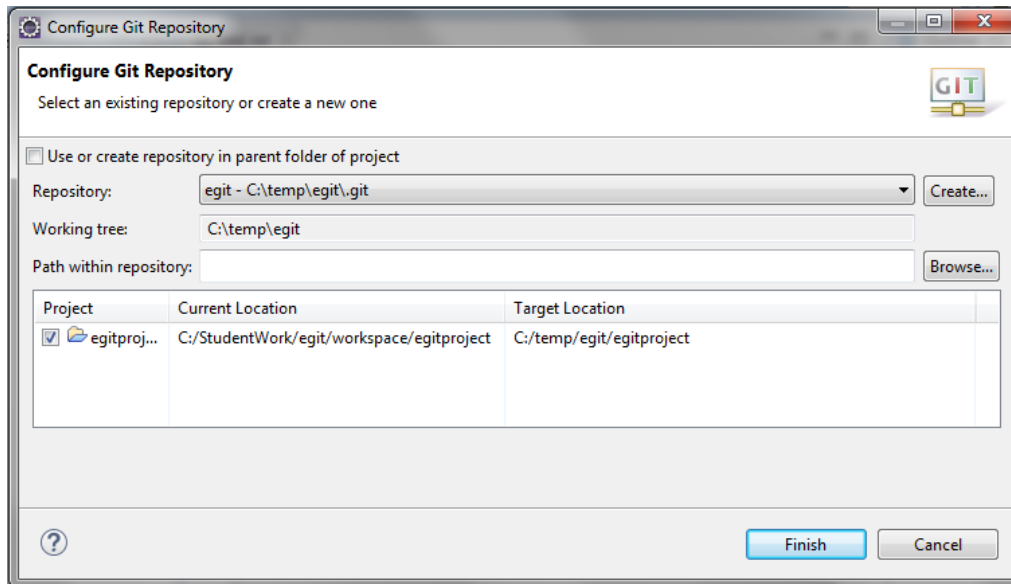
file1.txt:

```
"Hello there my friends"
```

Save the file.

Within the Project Explorer right click on the egitproject. Choose Team -> Share Project.

In the Repository pull down select our egit repository. Then press Finish



Right click on the Project again and choose Team -> Add to Index
 Right click on the Project again and choose Team -> Add to Index

in the git project folder: c:\temp\egit\egitproject

Open your git shell and do a dir in the project folder:

```
posh-git - egit [master]
Copyright (C) 2009 Microsoft Corporation. All rights reserved.
C:\Users\toddw\Documents\GitHub> cd C:\temp\egit\egitproject
C:\temp\egit\egitproject [master +2 ~0 -0]>
C:\temp\egit\egitproject [master +2 ~0 -0]>
C:\temp\egit\egitproject [master +2 ~0 -0]> dir

Directory: C:\temp\egit\egitproject

Mode                LastWriteTime         Length Name
----                -
-a---             6/19/2016   2:05 PM             215 .project
-a---             6/19/2016   2:06 PM              22 file1.txt

C:\temp\egit\egitproject [master +2 ~0 -0]>
```

The master branch of the repository is active, and it shows 2 new files that are not in the repository.

Commit the new file to the repository:

```
git commit -m "Initial file1 commit"
```

```
posh-git ~ egit [master]

Directory: C:\temp\egit\egitproject

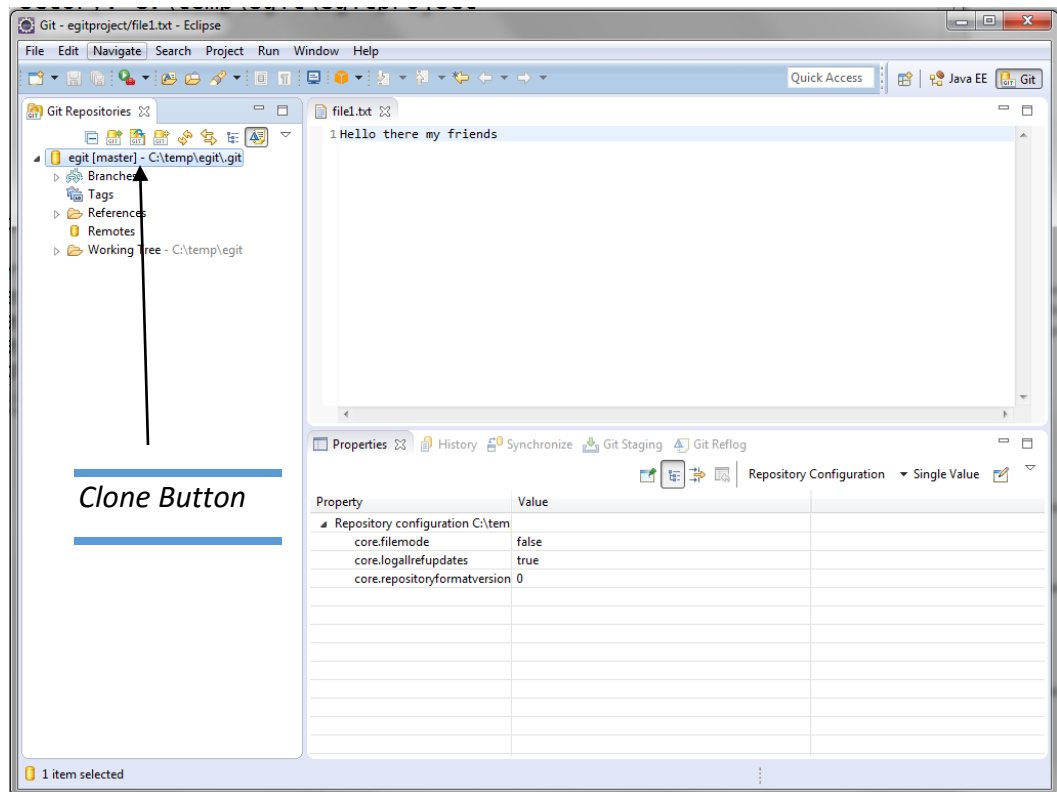
Mode                LastWriteTime         Length Name
----                -
-a---             6/19/2016   2:05 PM           215 .project
-a---             6/19/2016   2:06 PM           22 file1.txt

C:\temp\egit\egitproject [master +2 ~0 -0]> git commit -m "Initial file
[master (root-commit) 535b33c] Initial file1 commit
 2 files changed, 12 insertions(+)
 create mode 100644 egitproject/.project
 create mode 100644 egitproject/file1.txt
C:\temp\egit\egitproject [master]>
```

We easily could have committed in Eclipse (and we will..) but for this last example we added files to the index in Eclipse and then use the git command-line to do the commit.

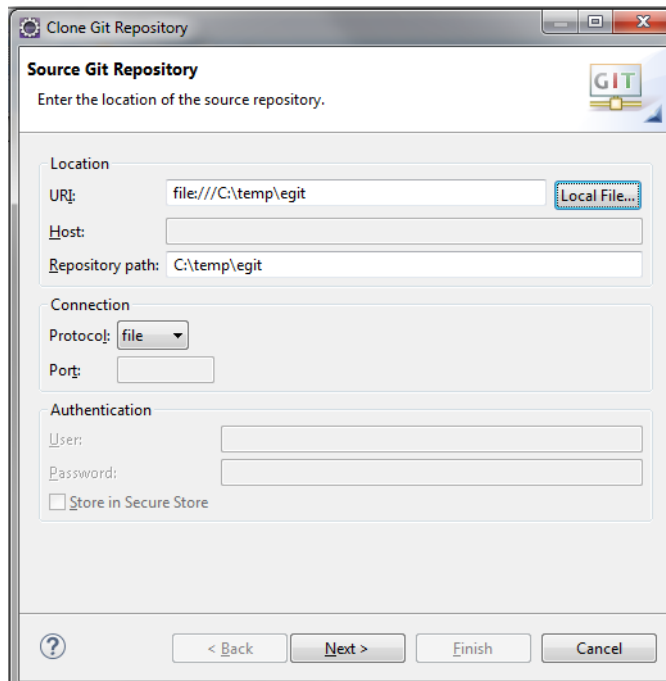
Step 2: Cloning a repository

Developers will usually clone a main repository (most likely from a remote site -- Local in this particular case) locally on their development platform. Changes will be made to their own repository and then occasionally pushed back to the main repository.



In your Eclipse Git perspective click on the Clone button immediately above the egit repository.

In the Source Git Repository choose the File button and then browse to the c:\temp\egit folder



Press the Next button to get the Branch Selection Press Next

In the Local Destination box type c:\temp\egitclone

We do not need to do another clone, but if you wanted to do a command line clone we would:

Run the command:

```
git clone c:\temp\egit c:\temp\egitclone2
```

This will create an entire clone of the main repository. The first argument to the git clone will be a url or local path.

Take a look in the new c:\temp\egitclone folder and you should see our files.

Step 3: Updating a file in your repository

Files in your repository are monitored for changes. In our lab this repository is distinct from the "Main" repository that is under c:\temp\egit

Back in Eclipse edit the contents of our file1.txt (the copy that is in the c:\temp\egit folder) to be:

```
Hello there my friends  
Version #2
```

Run the status command to see that file 1 is changed, but the change is not yet staged for a commit:

```
git status
```

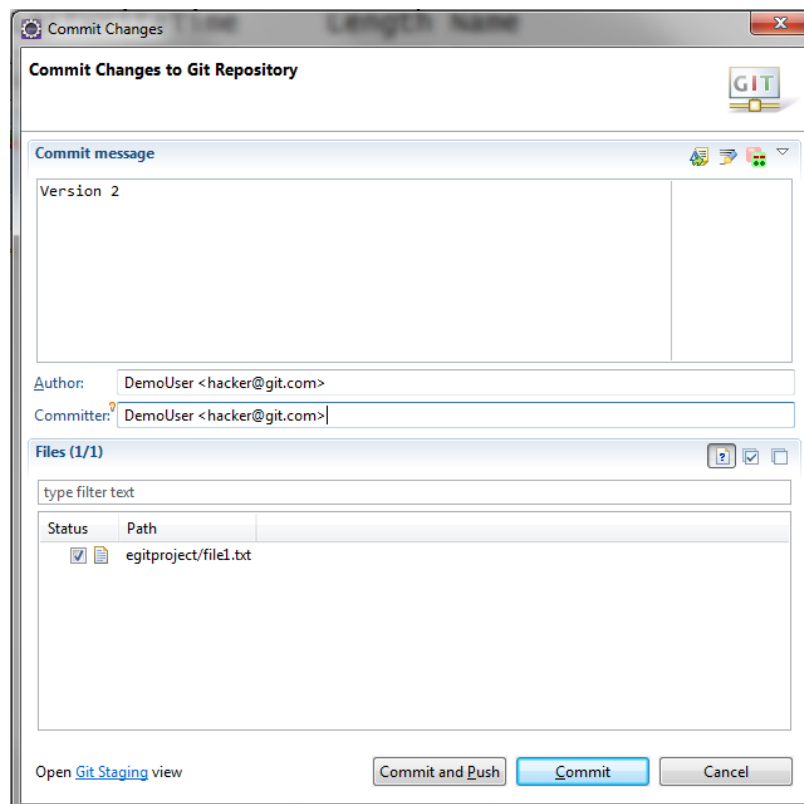
Add file1.txt to the stage

```
git add *.txt
```

(Alternatively we could have right clicked on our file (or the project) and added this to the index in Eclipse.

and then commit it to our repository - Lets use Eclipse this time.. :

Right click on our project in Eclipse. Team -> Commit



Press the commit button.

Exercise 3 – Working with branches Cmd Line



Overview

In this lab you will learn to branch a repository.

Objectives

At the conclusion of this exercise, you should be able to:

- ✓ Create a branch to your repository
- ✓ Compare versions of files within your repository
- ✓ Push changes to your branch

Step 1: Create a new branch



For the following task, we will continue working in our repository. We will branch our code make a change and then commit it to the branch.

Using the Git Shell, while inside the c:\temp\egit folder run the following command:

```
git branch experimental
```

The branch was created, but your prompt will still show that you are in the master branch. Change to the experimental branch

```
git checkout experimental
```

Your active branch is now experimental. Open our file1.txt and change the file to reflect version 3:

```
Hello there my friends  
Version #3
```

Step 2: Commit changes to the branch

If you use the git status command you will see that one file is changes, none are staged and nothing is there commit. Run the following command to stage the change to our branch:

```
git add file1.txt
```

Commit the change to our active "experimental" branch:

```
git commit -m "Version 3"
```

Step 3: Verify the different branches

Switch back to the master branch:

```
git checkout master
```

Look at the contents of the file. It should still be version 2.

Switch back to the experimental branch:

```
git checkout experimental
```

Look at the contents of the file. It should be version 3 again. The git diff command can tell us all kinds of interesting things. Relevant to us now is the difference between two branches of a particular file. Run the following command:

```
git diff master..experimental
```

You will see that the one line is removed and a replacement line was added. Change back to the master branch and then merge changes from experimental:

```
git checkout master  
git merge experimental
```

Exercise 4 – Working with branches GUI



Overview

In this lab you will learn to branch a repository and use the Eclipse Git tools.

Objectives

At the conclusion of this exercise, you should be able to:

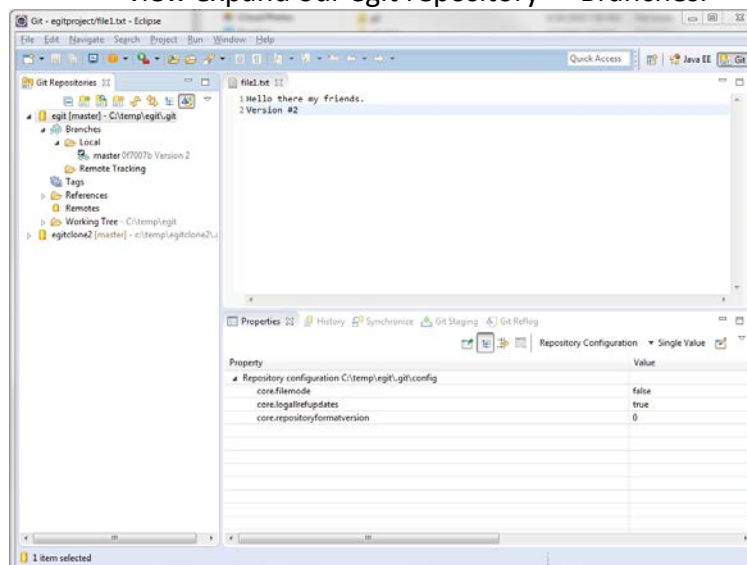
- ✓ Create a branch to your repository
- ✓ Compare versions of files within your repository
- ✓ Push changes to your branch

Step 1: Create a new branch

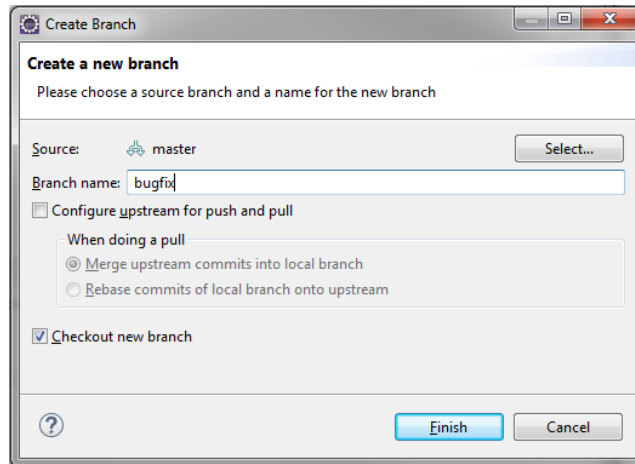
For the following task, we will continue working in our repository. We will branch our code make a change and then commit it to the branch.



In Eclipse Switch to the Git perspective. Using the Git Repositories view expand our egit repository -> Branches.



Right click on Branches and select Switch To -> New Branch
Name the new branch bugfix



The branch was created. You should see a small checkmark next to the bugfix branch in the Git Repository view.

Step 2: Commit changes to the branch

If you were to make a change now, Add it to the index and then commit the change, it would all happen in the bugfix branch.

Try It:

Make a change to our file

Add the file to the index

Commit the change.

Step 3: Verify the different branches

Switch back to the master branch:

Use the Git Perspective and the Git Repository view to checkout the master branch.

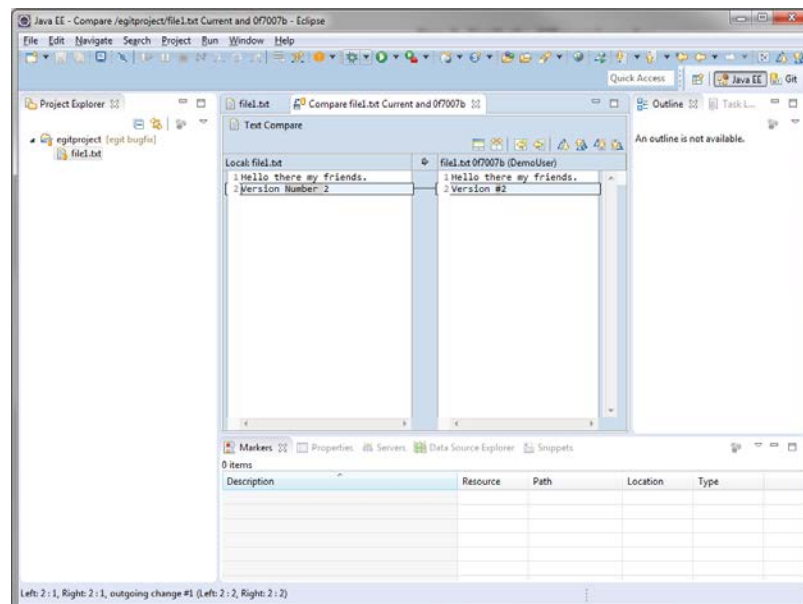
If you had the file open in your editor you should have seen the version switch back to the older revision.

Look at the contents of the file. It should be version 3 again.

The git diff command can tell us all kinds of interesting things. Relevant to us now is the difference between two branches of a particular file. The GUI EGit tool uses the standard Eclipse "Compare With" context menu.

While in the project explorer within eclipse, Right click on file1.txt choose Compare With -> Branch, Tag or Reference

Choose to compare with the Master branch (We are assuming you are in the bugfix" branch currently)



You will see that the change highlighted. The buttons in the top of the Text Compare pane will let you copy changes from one side to the other. -- Helpful for incorporating someone else's fixes into your code.

Exercise 5 – GitHub



Overview

In this exercise you will create a GitHub account and repository, then synchronize your project with GitHub

Objectives

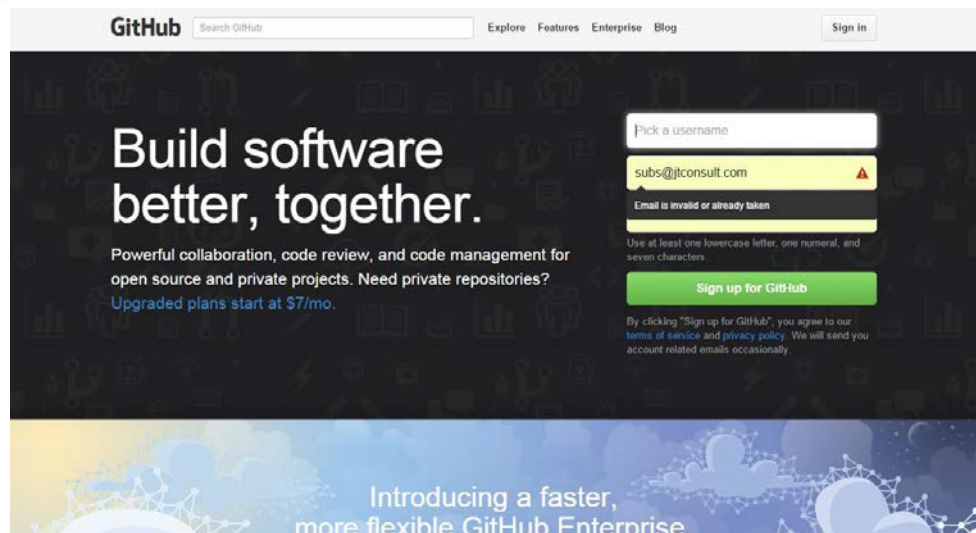
At the conclusion of this exercise, you should be able to:

- ✓ Create a Github repository
- ✓ synchronize your repository with Github



Step 1: Create a GitHub account

In a browser go to www.github.com and create an account.



Once you have your account login to the github web site.


Step 2: Change your local Git settings to match GitHub

Your local git settings need to be in synch with Github so your account will match. In your git shell within the c:\temp\gitclass project execute the following command, changing "Your Email" to .. your email.

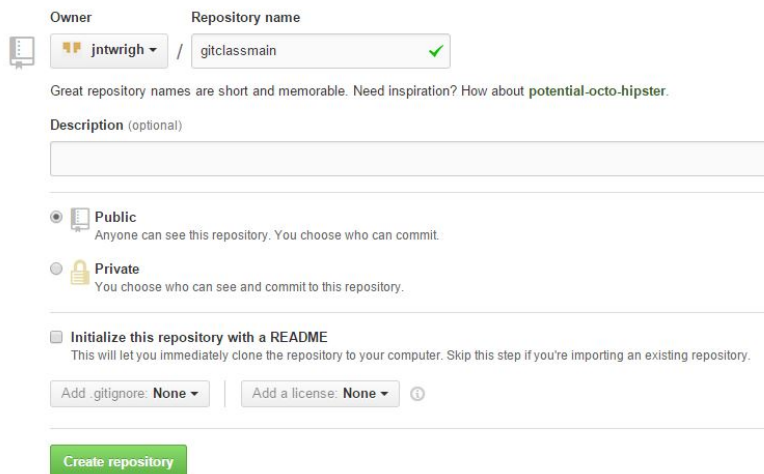
```
git config --global user.email "YOUR EMAIL"
```

Step 3: Create a GitHub Repository

Back in the browser go to: <https://github.com/> and log in if you are not already logged in.

Press the  (plus) button next to your username to create a new repository.

Name the repository gitclassmain -- leave the repository as Public and press the green Create repository button



Once you hit Create copy into your clipboard the link from the quick setup box. it will look something like:

```
https://github.com/USERNAME/gitclass.git
```

Step 4: Synchronize your local repository with GitHub

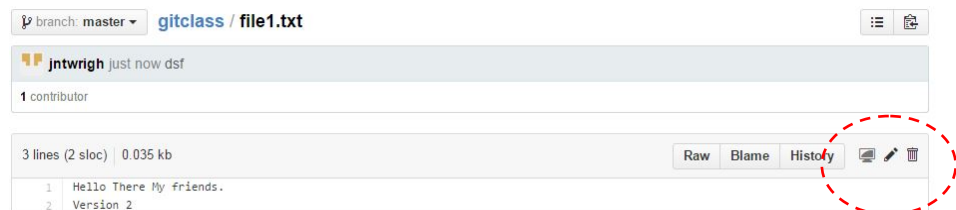
In your git shell within the c:\temp\gitclass project run the following command to push your project to a new remote repository: (Swap the URL with the URL you copied in the last step.

```
git remote add github https://github.com/USERNAME/gitclass.git
```

We have created a remote server reference named github that we can now push our project to. Run the following command to actually push your project up to github.

```
git push -u github master
```

In your web browser look at your github repository (You may have to refresh the screen, but you should see the file1.txt is up on github. Drill into file1.txt and view the contents.



Edit the file to say Edited on GitHub, then click on the Preview Changes link, add a version message and then commit the changes.

Back in your shell run the following command:

```
git pull github
```

Then view your file and you should see the changes from github.

Exercise 5 – Obtaining a repository



Overview

In this exercise, you will clone an existing repository and begin to make changes to it.

Objectives

At the conclusion of this exercise, you should be able to:

- ✓ Clone a network repository
- ✓ Extract content from git
- ✓ Verify the integrity of a local repository

Step 1: Create a local copy of a remote Git Repository



In your GIT shell change to c:\temp and run the following command to create a project folder:

```
mkdir c:\temp\gitclass
```

Change into that folder:

```
cd c:\temp\gitclass
```

Clone a public demo Github project with the following command:

```
git clone https://github.com/ajaxorg/demo-project.git
```

```
Administrator: C:\Windows\System32\WindowsPowerShell\v1.0\Powershell.exe
C:\tmp\gitclass> git clone https://github.com/ajaxorg/demo-project.git
Cloning into 'demo-project'...
remote: Counting objects: 71, done.
remote: Total 71 (delta 0), reused 0 (delta 0), pack-reused 71
Unpacking objects: 100% (71/71), done.
Checking connectivity... done.
C:\tmp\gitclass>
```

Step 2: Locate your objects

In your shell traverse into the demo-projects/html folder. We need to list the hash for the index.html so we can find it in the objects folder shortly. Run the following command from within the demo-projects/html folder:

```
git ls-files -s .\index.html
```

Make a note of the first 6-8 chars of the hash code. (Yours should be different from the screen)

```
posh-git - demo-project [master]
C:\tmp\gitclass\demo-project [master] > cd html
C:\tmp\gitclass\demo-project\html [master] > git ls-files -s .\index.html
100644 6ad3950fc62e2dad31de8e908b2fa4de75c7a210 0      index.html
C:\tmp\gitclass\demo-project\html [master] >
```

In your shell traverse into the demo-projects/.git/objects folder. You will find many folders with 2 letter names. These are the first two letters in the hash for the objects stored by git.

You will find many folders with 2 letter names. These are the first two letters in the hash for the objects stored by git. Switch into the folder that is named with the first two letters

of your hash value and look for the file with the remaining name.

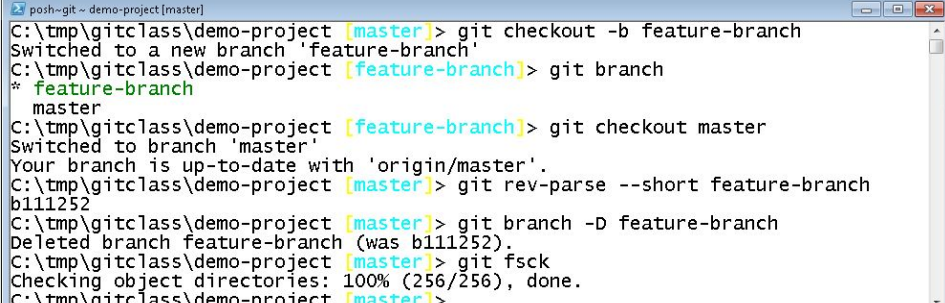
****** Just a note the `git ls-files` command will show you what is in the current index. The `git hash-object` command will show you the hash in the current tree, and as you edit the files they will be different.

Step 3: Create (and fix) a dangling branch

Run the following command to create a branch, retrieve the hash for it, delete it and then run the `fsck` to ensure the index is accurate.

While inside the demo-project folder run:

```
git checkout -b feature-branch
git branch
git checkout master
git rev-parse --short feature-branch
git branch -D feature-branch
git fsck
```



```
posh-git - demo-project [master]
C:\tmp\gitclass\demo-project [master]> git checkout -b feature-branch
Switched to a new branch 'feature-branch'
C:\tmp\gitclass\demo-project [feature-branch]> git branch
* feature-branch
  master
C:\tmp\gitclass\demo-project [feature-branch]> git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
C:\tmp\gitclass\demo-project [master]> git rev-parse --short feature-branch
b111252
C:\tmp\gitclass\demo-project [master]> git branch -D feature-branch
Deleted branch feature-branch (was b111252).
C:\tmp\gitclass\demo-project [master]> git fsck
Checking object directories: 100% (256/256), done.
C:\tmp\gitclass\demo-project [master]>
```

Step 4: Create and remove "dangerous" content

It happens that developers accidentally commit content to a repo that should not have been stored in the repo. Once it is in the repo the file will be persisted across commits and cleaning it out can be very difficult.

Add a new text file in the `html` folder of our demo-project called `passwords.txt`:

```
db.url=192.168.0.1:1521
db.username=Scott
db.password=tiger
```

Add it to the index folder and then commit it to the project:

```
git add html/passwords.txt
git commit -m "adding a dangerous file"
```

Edit a few files and commit them. -- Do at least 3 minor changes to the index.html file and commit each one individually.

Run the git log command to view your commit history:

```
git log
```

Remove the passwords.txt from all commits of the current branch:

```
git filter-branch --tree-filter 'rm -rf html/passwords.txt' HEAD
```

Exercise 6 – Git Configuration



Overview

In this Lab you will manage your Git configuration and debug common problems when using Git.

Objectives

At the conclusion of this exercise, you should be able to:

- ✓ Locate and edit the global .gitconfig
- ✓ Locate and edit the project local Git configuration
- ✓ Find files within your repository
- ✓ Track bugs introduced to the repository

Step 1: Locate your global git configuration file



Git is configured by a set of variables. The set of variables that GIT uses are located at <https://git-scm.com/book/en/v2/Git-Internals-Environment-Variables> . The particular "Active" settings are dictated by project specific and global git configurations. Project configurations override the global settings.

1. Locate your global Git configuration file (.gitconfig) :
 - a. Windows users look in your c:\users\username\ directory for the .gitconfig
 - b. MAC/Linux users look in your \$HOME/ for .gitconfig
 - c. Run the command:

```
git config --global user.name "John Doe"
```
 - d. run the following command to view the global configuration - You should see the new Global user name setting:

```
git config --global --edit
```

- e. Edit the user.name property to be your name again.
2. The repository specific settings are stored in the repository folder. Git repository settings override the global settings. Open the demo-project/.git/config file
- a. What is the remote "origin" url? - What is the purpose of the origin repository?
 - b. Within the repository folder run the following command:

```
git config user.name "John Doe"
```

- c. Open the project config file again and you should see your new "name"
- d. Edit the user.name property to be your name again.

Exercise 7 – Git Commands



Overview

In this Lab you will utilize more advanced git commands.

Objectives

At the conclusion of this exercise, you should be able to:

- ✓ Create aliases to common git commands
- ✓ Find files effectively in git
- ✓ Locate when changes were introduced
- ✓ undo changes to a project
- ✓ stash a uncommitted change

Step 1: Define git Aliases



Aliases make working with GIT much easier. In this section you will list, create and delete aliases to git commands.

1. List all existing git aliases by running the following command:

```
git config -e
```

Look for the aliases

2. Create the following git aliases

```
git config --global alias.alias "config --get-regexp ^alias\."
```

Then run `git alias`

```
git config --global alias.co checkout
git config --global alias.ci commit
git config --global alias.df diff
```

3. Delete the following aliases with the following command:
`git config --global --unset alias.df`

Step 2: Find content on the repository

Git is a distributed filesystem. It can get complex and large. Many different users all making commits and adding content can lead to lost files at times.

1. Add additional content to our repository:

Make sure you are in the master branch of your repository:

```
git checkout master
```

copy the contents of the 3-2.1 folder to the demo-project folder. From within the demo-project folder run: Override files if necessary

(win)

```
copy c:\studentfiles\3-2.1\* .
```

(mac)

```
copy /studentfiles/3-2.1/* .
```

```
git add --all
```

```
git commit -m "v3-2.1"
```

2. Add more content to our repository:

copy the contents of the 3-2.2 folder to the demo-project folder. From within the demo-project folder run: Override files if necessary

(win)

```
copy c:\studentfiles\3-2.2\* .
```

(mac)


```
copy /studentfiles/3-2.2/* .
```

```
git add --all
```

```
git commit -m "v3-2.2"
```

3. Add more content to our repository:

copy the contents of the 3-2.3 folder to the demo-project folder. From within the demo-project folder run: Override files if necessary

(win)

```
copy c:\studentfiles\3-2.3\* .
```

(mac)

```
copy /studentfiles/3-2.3/* .
```

```
git add --all
```

```
git commit -m "v3-2.3"
```

4. Assuming we had a much more complex hierarchy with millions and millions of very valuable files, locating a particular file could potentially be difficult. We do recall that somewhere in our vast library we are using goodbye.js. Run the following command to locate the offending file:

```
git config --global grep.lineNumber true
```

```
git config --global alias.g "grep --break --heading --line-number"
```

```
git grep -e 'goodbye'
```

```
git branch chap03
```

```
git grep -e 'goo\w*' chap03 -- '*.html'
```

5. Someone introduced a BUG into the project.. I need to blame someone. Use the git blame utility to find out who edited the hello.js file and introduced the bug into it.

Switch into the html/js folder and run the following command:

```
git blame hello.js
```

Who edited it last? When was it edited? Make a note of the hash for the BUG
BUG BUG line: _____

What was the hash of the line before the BUG? _____

6. The BUG was found, but we still want to know how far back it was inserted - how many commits have occurred since the introduction of the BUG. Insert the hash before the BUG in the second command.

Switch back to the root of your repo:

```
git bisect start
git bisect good %HASH_BEFORE_BUG%
git bisect bad
```

Do you see that v3-2.2 was the change?

Run the bisect reset command to start over:

```
git bisect restart
```

Can you edit the html/js/hello.js and change the BUG line to something else?
Commit your change.

Make 3 more changes to the hello.js, committing after each one.

Re-run the above commands

When you are done be sure to leave the bisect:

```
git bisect reset
```

Step 3: Stash changes

While working on a project you may need to switch to work on a different branch. Without losing your changes you want to change "views" and desire to return to that point in time later.

1. Switch to a branch and make changes:

Switch to the chap03 branch :

```
git checkout chap03
```

Open html/hello.html and make changes to the file.

Change the message to be "Hello from the chap03 branch"

2. A manager just walked in your office in a really bad mood. There is a very urgent update needed.

Switch to the master branch and correct a file:

```
git checkout master
```

Edit html/count.html to have a "Thank You message"

Add the change and commit the change:

```
git add *  
git commit -m "Lab 3.3"
```

3. Return to the chap03 branch and look at the html/hello.html file.

```
git checkout chap03
```

Is your Hello from the chap03 branch message still there?

Why not?

4. Re-edit the html/hello.html file. Put our change back in there.

stash you change:

```
git stash
```

switch back to the master branch:

```
git checkout master
```

make a change, commit it, then return to the chap03 branch.

```
edit count.html  
git add *  
git commit -m "stash Lab"  
git checkout chap03
```

5. Apply the stash changes:

```
git stash apply
```

Step 4: Creating a patch

A patch is a set of changes that will be applied to bring a set of files up to the current state of the current branch.

1. Create a Patch from the chap03 branch:

Switch to the chap03 branch :

```
git checkout chap03
```

Create a patch:

```
git format-patch master > chap03.patch
```

2. Verify that the patch will work:

```
git checkout master  
git apply --stat chap03.patch
```

Step 5: Rebasing a repository

When a branch of a repository deviates from the master branch and needs to be re-incorporated back into the master branch we may desire to bring changes that were made to the master branch into the current branch. Essentially, we are changing the base commit that the branch started with.

1. Ensure that you have at least a branch named chap03 and the master branch:

```
git branch
```

2. Add a new file to the master branch:

```
git checkout master
notepad brandnewfile.txt
git add *
git commit -m "brandnewfile added"
```

3. Verify that the file is not in the chap03 branch:

```
git checkout chap03
dir / ls
```

4. Rebase your chap03 project to the current master commit level:

```
git rebase master
dir / ls
```

5. Is the new file available in the chap03 branch?
ans: -- It should be