GIT

Exercises

Exercise 1 – GitHub Setup

Overview

In this exercise, you will setup Github for Windows (or Mac).

Objectives

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

- ✓ Install GitHub for Windows (or Mac)
- ✓ Use the Git command line
- ✓ Create an initial repository

Step 1: Preparation

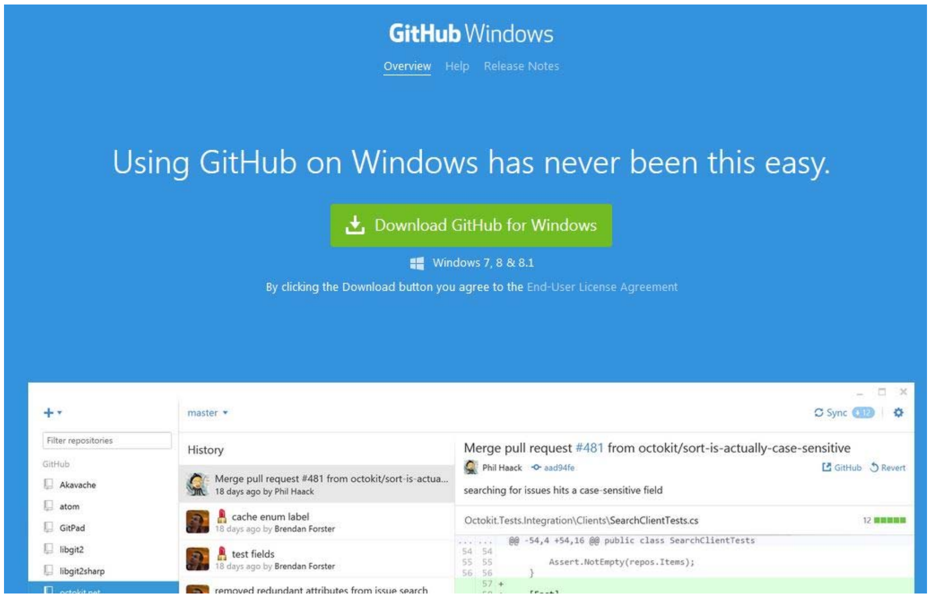
In this exercise, you will obtain and install a GitHub environment. Begin by downloading the GitHub implementation for your platform.

Either Windows:

https://windows.github.com/

Or Mac:

https://mac.github.com/


GitHub Windows

Overview Help Release Notes

Using GitHub on Windows has never been this easy.

Download GitHub for Windows

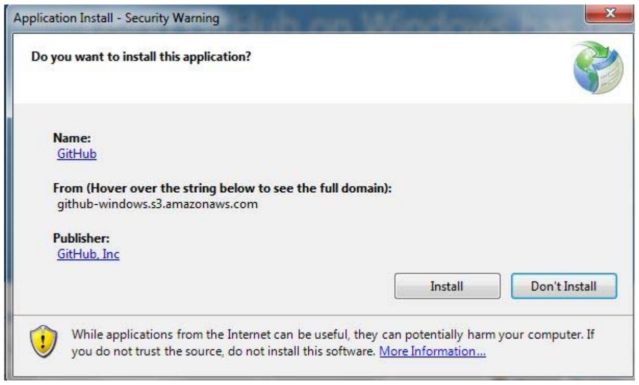
Windows 7, 8 & 8.1

By clicking the Download button you agree to the End-User License Agreement

master
Sync
Filter repositories
GitHub
Akavache
atom
GitPad
libgit2
libgit2sharp
History
Merge pull request #481 from octokit/sort-is-actua...
18 days ago by Phil Haack
cache enum label
18 days ago by Brendan Forster
test fields
18 days ago by Brendan Forster
removed redundant attributes from issue search
Merge pull request #481 from octokit/sort-is-actually-case-sensitive
Phil Haack
aad94fe
GitHub
Revert
searching for issues hits a case-sensitive field
Octokit.Tests.Integration\Clients\SearchClientTests.cs

Step 2: Install GitHub

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


Application Install - Security Warning

Do you want to install this application?

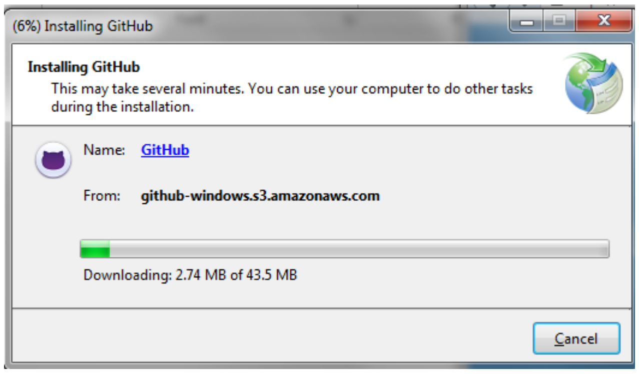
Name:
GitHub

From (Hover over the string below to see the full domain):
github-windows.s3.amazonaws.com

Publisher:
GitHub, Inc

Install Don't Install

While applications from the Internet can be useful, they can potentially harm your computer. If you do not trust the source, do not install this software. More Information...


(6%) Installing GitHub

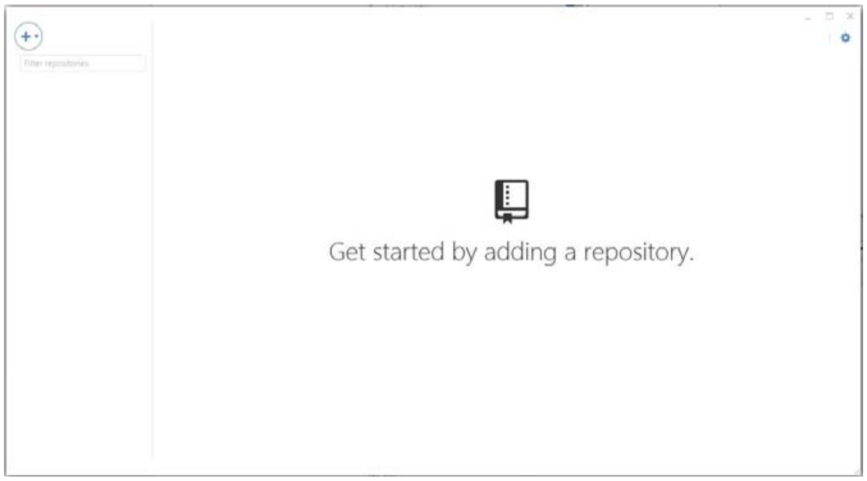
Installing GitHub
This may take several minutes. You can use your computer to do other tasks during the installation.

Name: GitHub

From: **github-windows.s3.amazonaws.com**

Downloading: 2.74 MB of 43.5 MB

Cancel



Close the GitHub for Windows (or Mac)

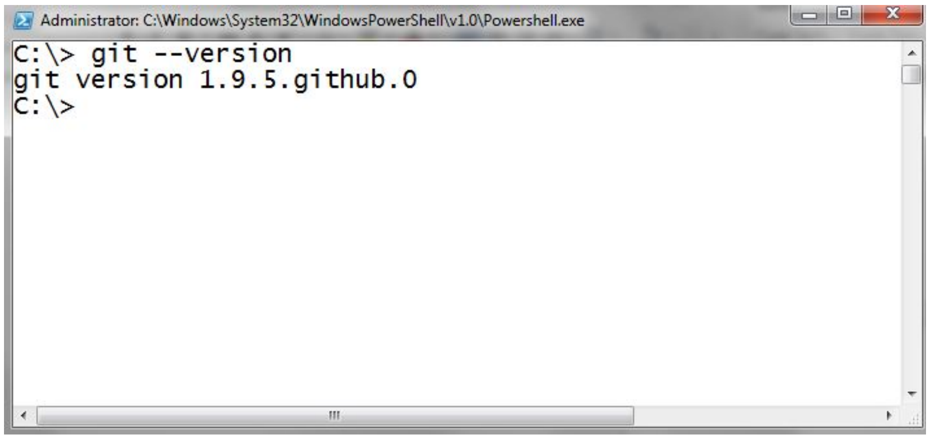
Step 3: Use Git Shell

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
```



Administrator: C:\Windows\System32\WindowsPowerShell\v1.0\Powershell.exe

```
C:\> git --version
git version 1.9.5.github.0
C:\>
```

Step 4: 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 5: Create a local Git Repository

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

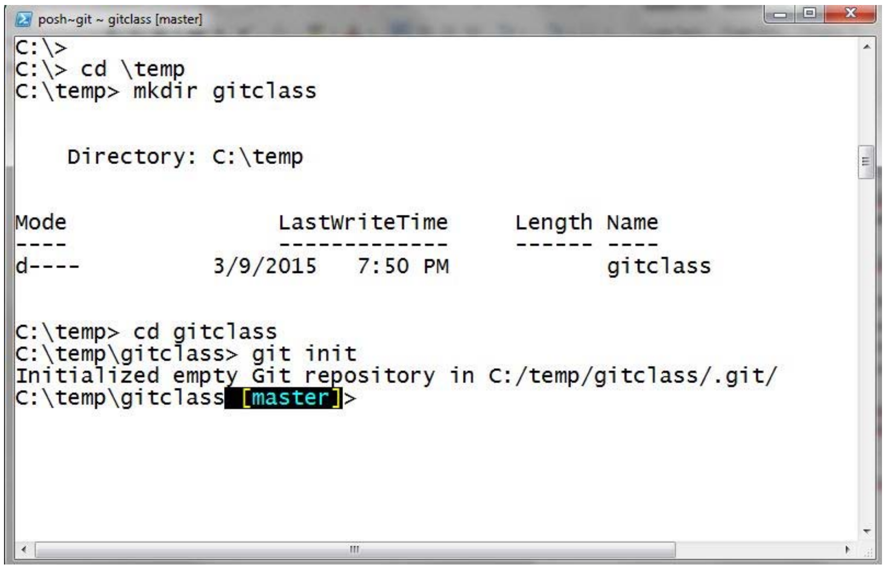
```
mkdir c:\temp\gitclass
```

Change into that folder:

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

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

```
git init
```



```
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.

Exercise 2 – Managing Files in Git

Overview

In this Lab you will add and manage files in your "main – master" 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

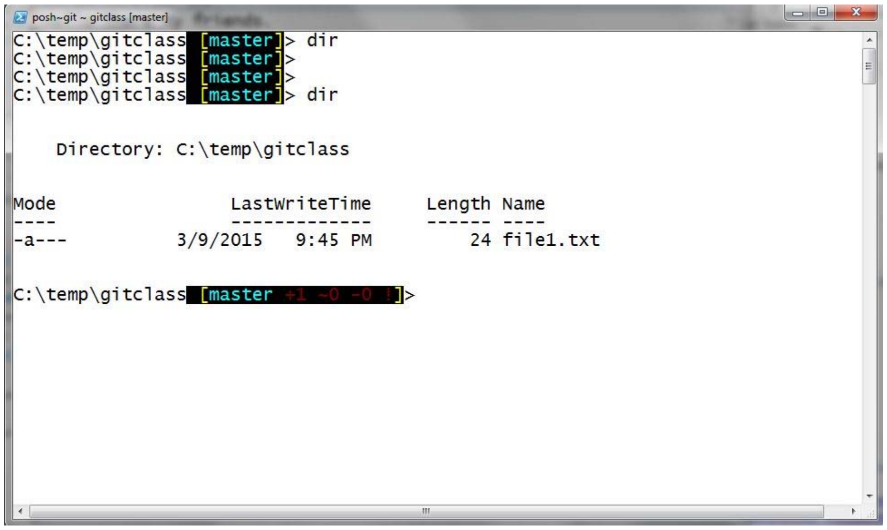
Open a text editor and create a file named file1.txt with a simple text message inside it :

file1.txt:

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

Save the file in the git project folder: `c:\temp\gitclass`

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



```
C:\temp\gitclass [master]> dir
C:\temp\gitclass [master]>
C:\temp\gitclass [master]>
C:\temp\gitclass [master]> dir


    Directory: C:\temp\gitclass


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---          3/9/2015   9:45 PM         24 file1.txt


C:\temp\gitclass [master +1 ~0 -0 !]>
```

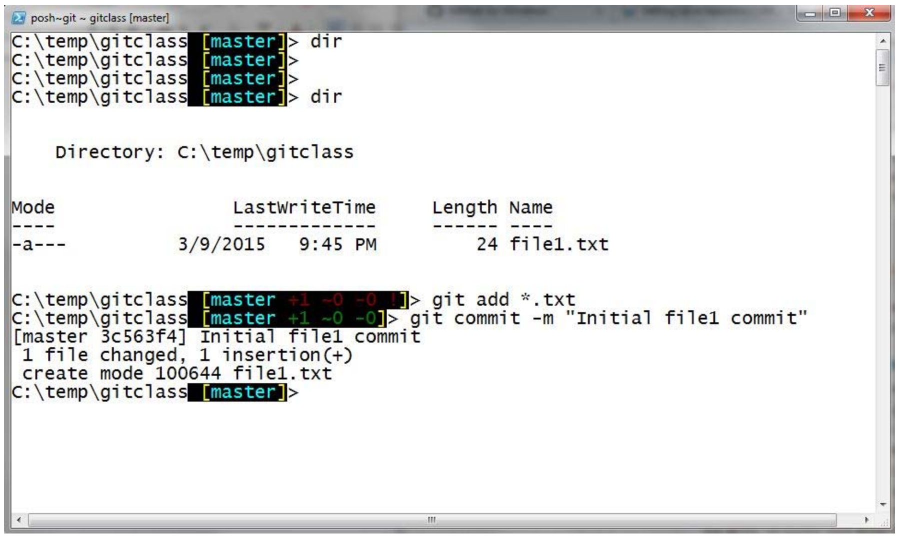
The master branch of the repository is active, and it shows 1 new file that is not in the repository.

Add the .txt file to the control of the repository:

```
git add *.txt
```

Commit the new file to the repository:

```
git commit –m "Initial file1 commit"
```



```
C:\temp\gitclass [master]> dir
C:\temp\gitclass [master]>
C:\temp\gitclass [master]>
C:\temp\gitclass [master]> dir


    Directory: C:\temp\gitclass


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---          3/9/2015   9:45 PM         24 file1.txt


C:\temp\gitclass [master +1 ~0 -0 !]> git add *.txt
C:\temp\gitclass [master +1 ~0 -0]> git commit -m "Initial file1 commit"
[master 3c563f4] Initial file1 commit
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt
C:\temp\gitclass [master]>
```

Step 2: Cloning a repository

Developers will usually clone a main repository locally on their development platform. Changes will be made to their own repository and then occasionally pushed back to the main repository.

Open a new git shell window and change to the `c:\temp` folder

Run the command:

```
git clone c:\temp\gitclass c:\temp\gitclassclone
```

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\gitclassclone folder and you should see our file.

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\gitclass

Edit the contents of our copy of the file1.txt (the copy that is in the c:\temp\gitclassclone 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
```

and then commit it to our repository:

```
git commit –m "Version 2"
```

Exercise 3 – Working with branches

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 "developer" copy of the repository. We will branch our code make a change and then commit it to the branch.

Within the c:\temp\gitclassclone project folder run the following command:

```
git branch version3
```

The branch was created, but your prompt will still show that you are in the master branch. Change to the version2 branch

```
git checkout version3
```

Your active branch is now version3. 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 "version3" 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 version3 branch:

```
git checkout version3
```

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. Relevent to us now is the difference between two branches of a particular file. Run the following command:

```
git diff master..version3
```

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 version3:

```
git checkout master
git merge version3
```

Exercise 4 – GitHub

Overview

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

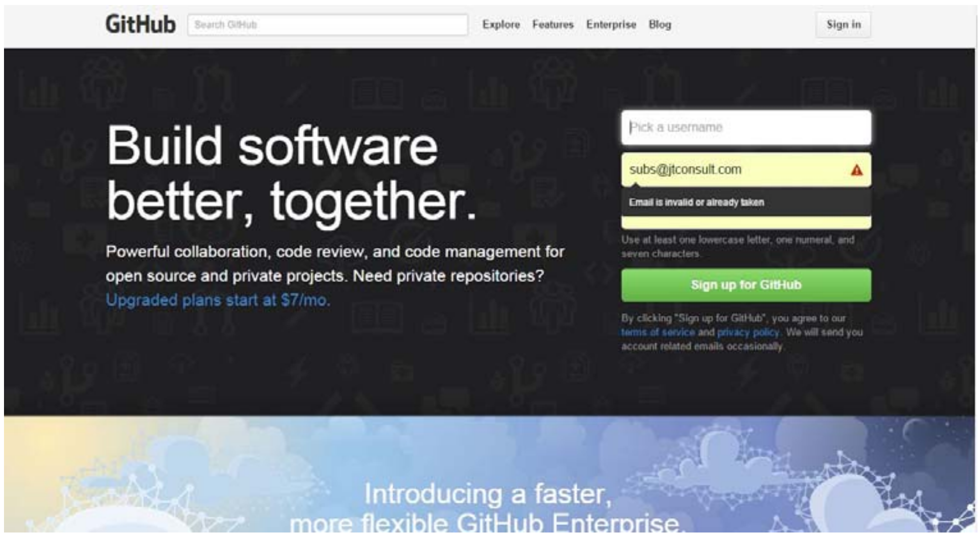
Objectives

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.



Build software better, together.

Powerful collaboration, code review, and code management for open source and private projects. Need private repositories? Upgraded plans start at $7/mo.

Pick a username

subs@jtconsult.com

Email is invalid or already taken

Use at least one lowercase letter, one numeral, and seven characters.

By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We will send you account related emails occasionally.

Introducing a faster, more flexible GitHub Enterprise.

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.

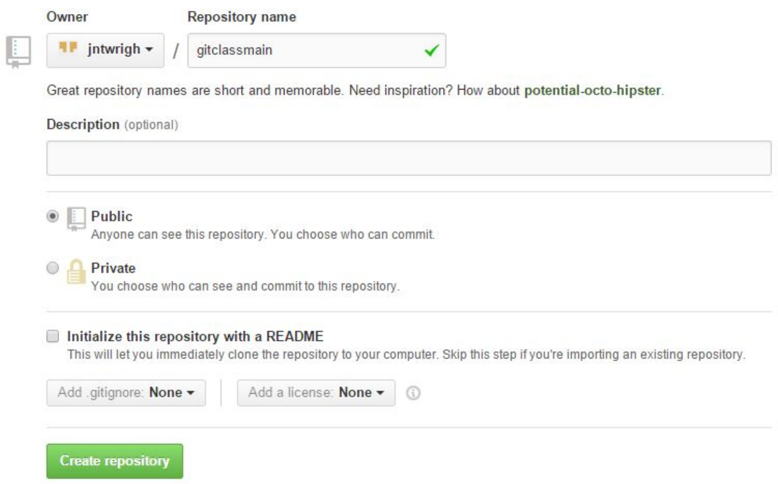
```
git config --global user.email "YOUR EMAIL"
```

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



Owner: jntwrigh / Repository name: gitclassmain ✓

Great repository names are short and memorable. Need inspiration? How about potential-octo-hipster.

Description (optional)

[✓ Public] Anyone can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾ | Add a license: None ▾

Create repository

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

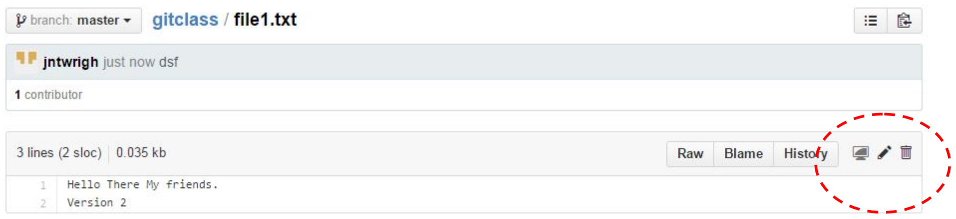
In your git shell within the c:\temp\gitclass project run the following command to push your project to a new remote repository: (Swap the URL with the URL you copied in the last step.

```
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.



```
Hello There My friends.
Version 2
```

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:

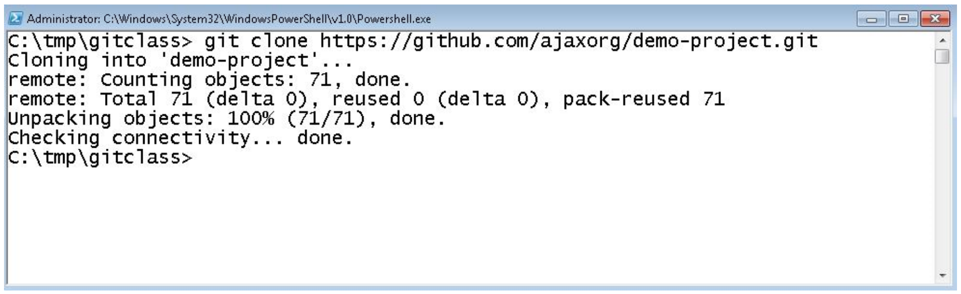
```
mkdir c:\temp\gitclass
```

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

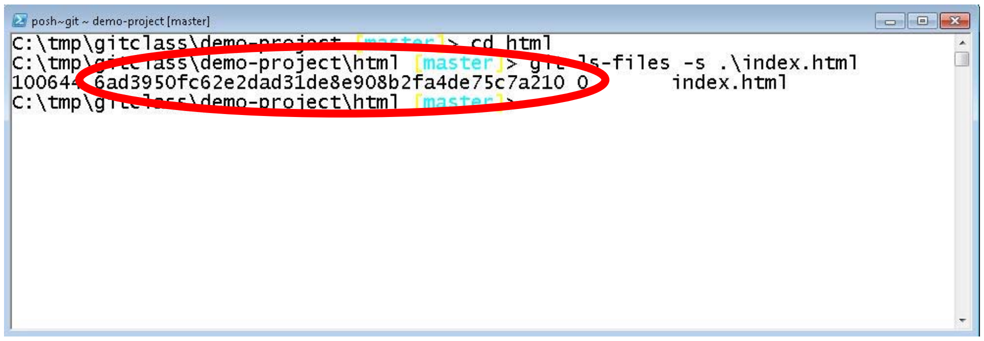
```
C:\tmp\gitclass> git clone https://github.com/ajaxorg/demo-project.git
Cloning into 'demo-project'...
remote: Counting objects: 71, done.
remote: Total 71 (delta 0), reused 0 (delta 0), pack-reused 71
Unpacking objects: 100% (71/71), done.
Checking connectivity... done.
C:\tmp\gitclass>
```

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)



```
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.

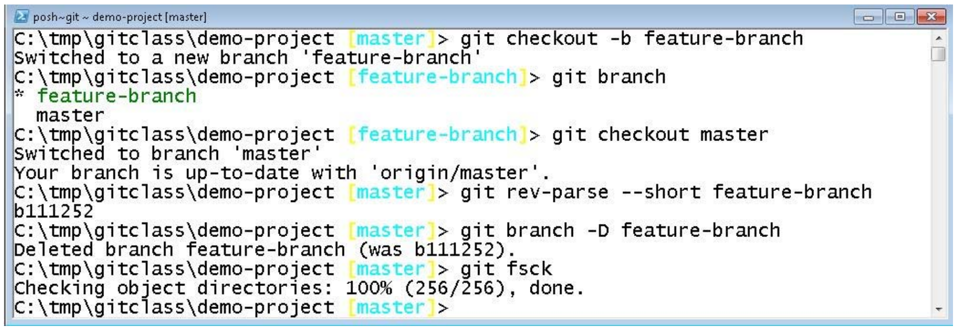
** Just a note the `git ls-files` command will show you what is in the current index. The `git hash-object` command will show you the hash in the current tree, and as you edit the files they will be different.

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
```



```
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 net 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