TortoiseGIT / GIT Tutorial:
Hosting a dedicated server with auto commit periodically on Windows 7 and Windows 8

2nd Edition

Abstract

This is a tutorial on how to host a dedicated gaming server on Windows 7 and Windows 8 using TortoiseGIT and/or GIT, whichever you prefer.

The intended audience is for those who runs a dedicated server that has an ever changing world map, for example, a Minecraft server, and for those who runs a dedicated server with constant changes to some files, such as leaderboards, player save data, etc.

Table of Contents

Changes from the previous edition

+ Added missing source code of the batch program script.
+ Added a warning near the end of the book.
+ Added a new referral URL.

Setup and Installation

The first thing you will need to do is to install the following softwares if you do not have them at this point of time.

- TortoiseGIT or GIT (Required)
- msysgit (Required)

I recommend with starting off with TortoiseGIT, as it has an easier learning curve than GIT. For advanced users, feel free to use GIT, as they both have common functionalities.

First, open up your preferred web browser and navigate to the TortoiseGIT project site (Figure 1). The URL is given below:

http://code.google.com/p/tortoisegit/

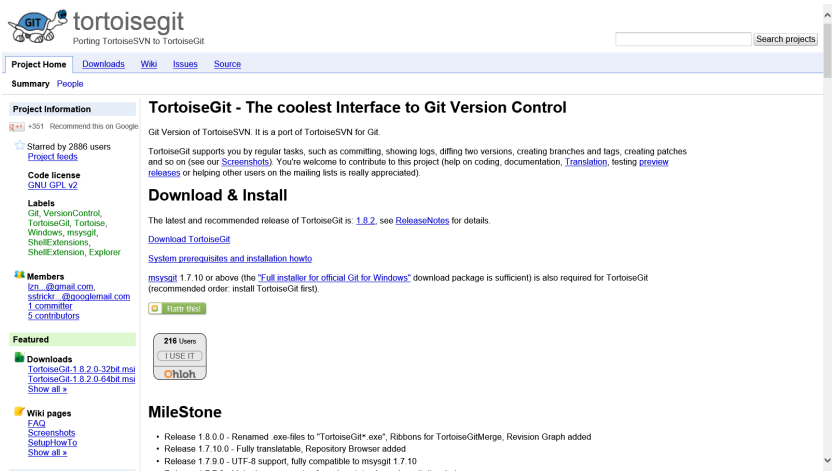
Click on the Downloads tab near the top of the project home page, and we should see two versions of TortoiseGIT and a list of language packs (Figure 2). If you are using an x86 operating system, please use the 32-bit version; otherwise, if you are using an x64 operating system, please use the 64-bit version. Both of these versions are not entirely the same. For more information, you may look it up on Wikipedia. In my case, I am using Windows 8 64-bit, so there will be minor differences from your operating system with mine in this tutorial.

If your native language is not English, you may select the correct version with your preferred language.

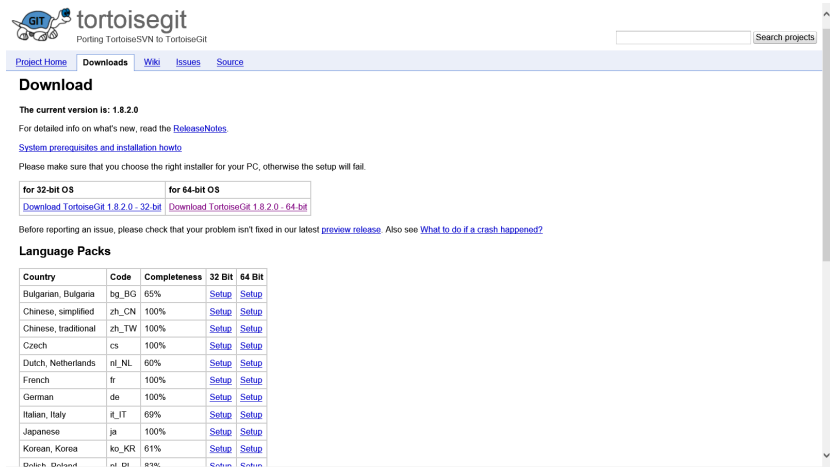
Once the setup files have finished downloading, use the File Explorer to navigate to the setup files and proceed to install them.

When TortoiseGIT has completed its installation, you may be prompted to install msysgit, which is the core part of GIT and provides GIT functions for TortoiseGIT and GIT on the Windows operating system. The URL for msysgit is given below, which will navigate you to its project site (Figure 3):

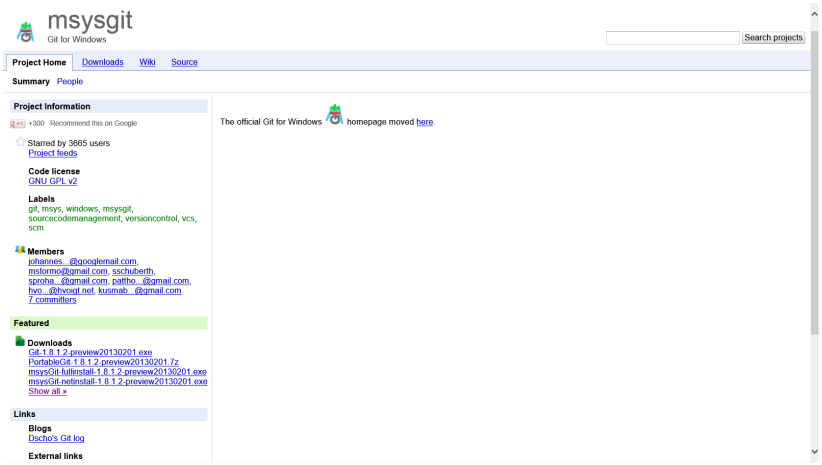
http://code.google.com/p/msysgit/



(Figure 1): TortoiseGIT project home page.



(Figure 2): The Downloads tab for TortoiseGIT.



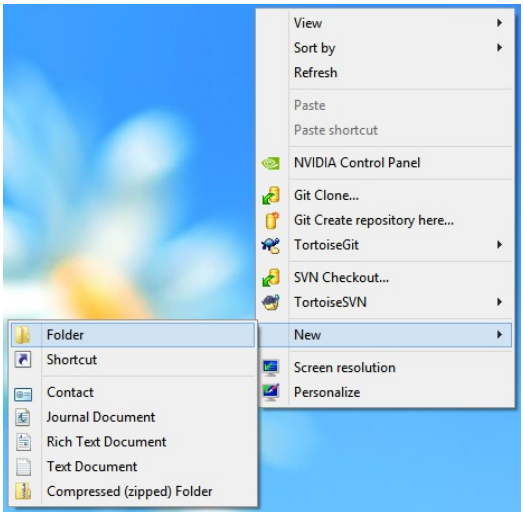
(Figure 3): msysgit project home page.

Once all TortoiseGIT and msysgit installations have been completed, the next thing to do is to determine where to create a repository folder for your dedicated server data. For more information about using GIT, you may visit the following URL:

http://www.vogella.com/articles/Git/article.html

Instructions

I will use the Desktop as my location for creating my repository for the purposes of this tutorial. To start, right click anywhere on the Desktop, in the context menu, hover over “New”, and click on “Folder” (Figure 4). Note that I have TortoiseSVN installed. Please ignore that for the time being.

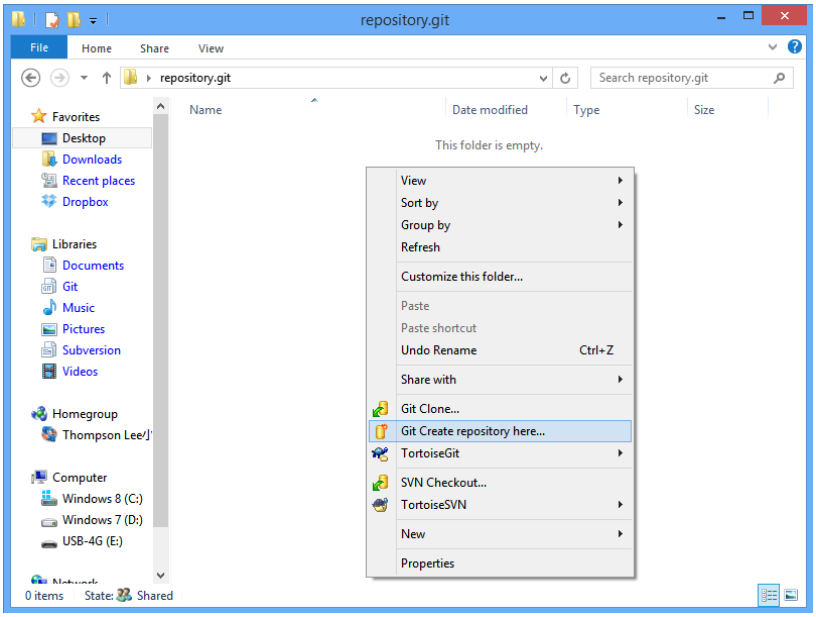


(Figure 4): The context menu.

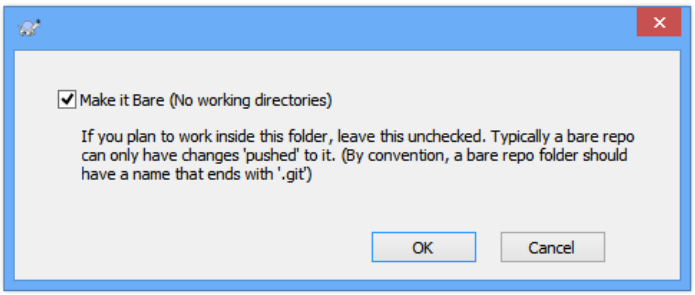
After a new folder has been created, rename the folder so that there is ".git" suffix at the end of the name. This is the GIT convention, and it's recommended to use this naming scheme. I have named mine, "repository.git", for example (Figure 5). This will be your repository folder.

(Figure 5): repository.git folder.

Open the "repository.git" folder, and right click anywhere in the empty space to open up the context menu. Click on "GIT Create repository here...", and a prompt will display (Figure 6). Enable "Make it bare (No working directories)", and finally click on OK (Figure 7). The "repository.git" folder should then be populated with files and folders. You may ignore the files and folders.



(Figure 6): The context menu (again).

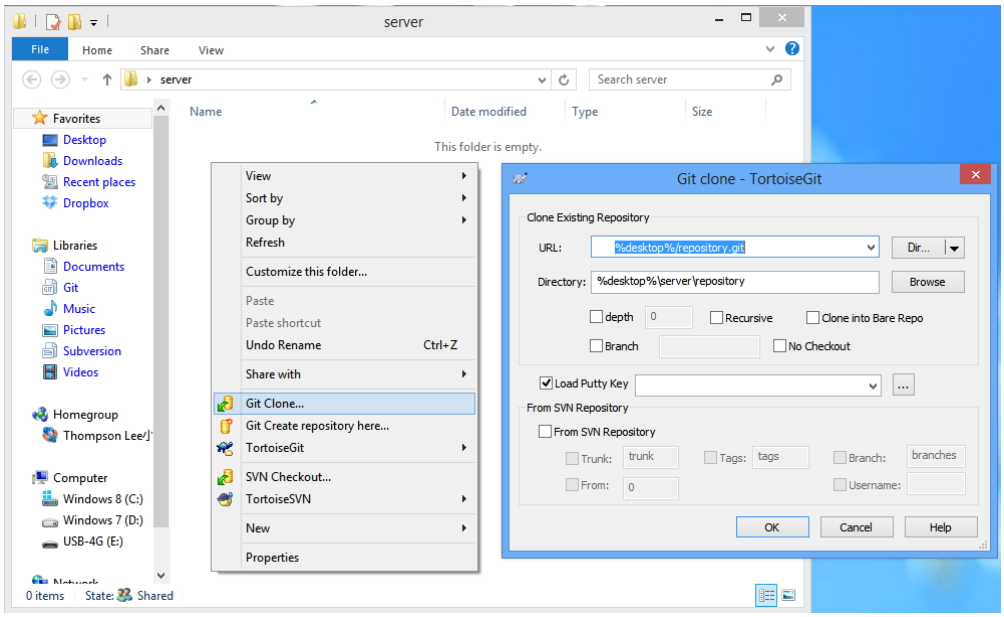


(Figure 7): The dialog prompt.

Going back to Desktop, create another new folder and name this folder to anything you want. I named mine to “server”. This folder shall then become the directory that you are to put your server data files in and commit to your repository folder. This will be your commit folder.

Open up the newly created folder you had just made, and right click anywhere again to bring up the context menu. Click on “GIT Clone...” to bring up the Clone dialog (Figure 8). For the URL, click on the “Dir...” button and find the folder, either the “repository.git” or the folder ending with the “.git” suffix that you had just created, and press OK. For the Directory, make sure that it is pointing towards the directory folder that you had just right clicked on “GIT Clone”, by browsing to the location. You may ignore the rest of the options given, and press OK once you’re done.

If you happened to spot a new folder directory inside your commit folder, and that new folder’s name is the same as your repository folder, but without the “.git” suffix, then what you did was you created a new commit folder nested in your “originally planned commit folder”. Your “originally planned commit folder” is no longer your commit folder, but rather it is just a file directory.



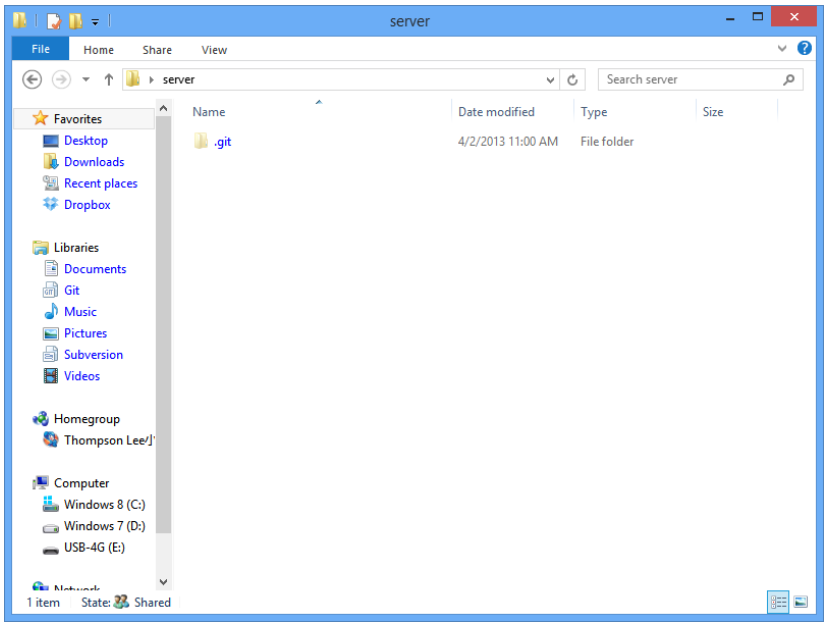
(Figure 8): The context menu and the GIT clone dialog.

If you have “Show hidden folders” option enabled, when you browse your commit folder, you should see a “.git” folder (Figure 9). This “.git” folder serves as a bookmarker for your local changes. You may ignore the “.git” folder, and continue with the tutorial.

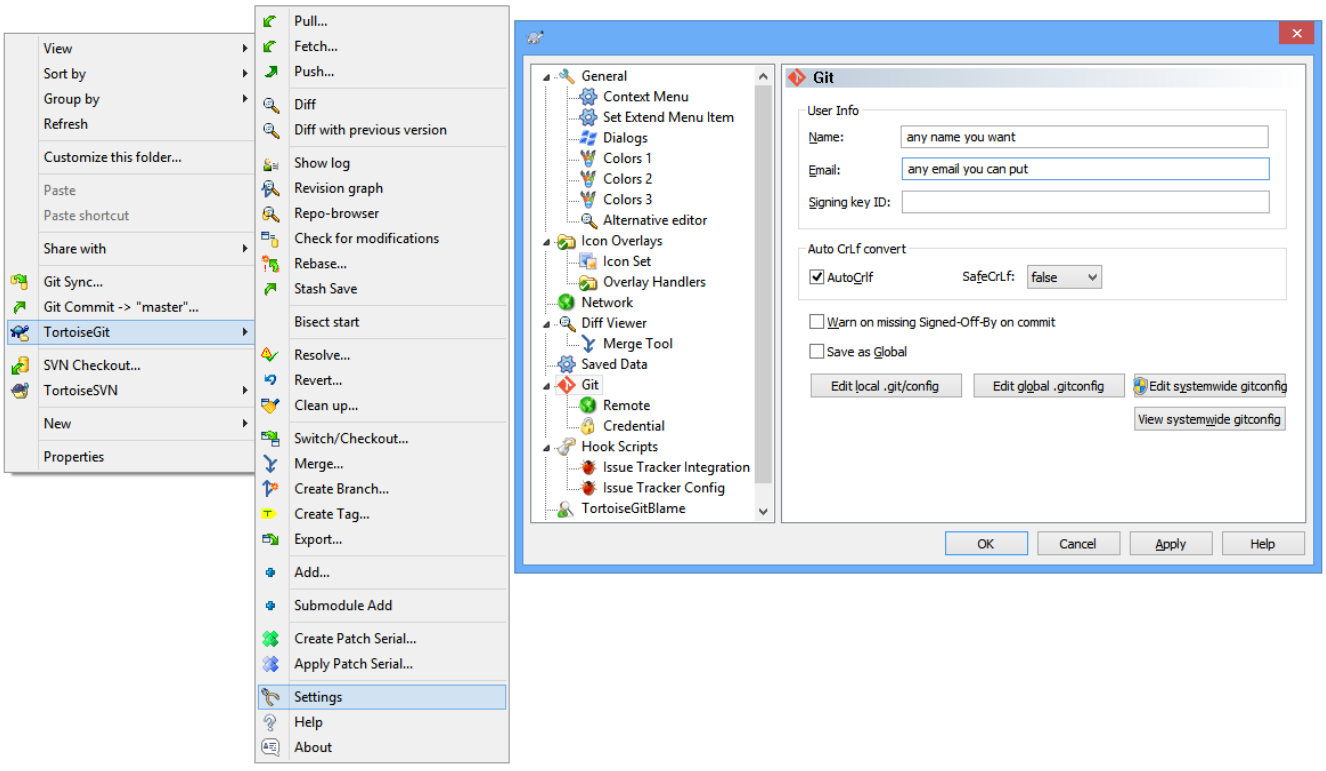
The next important step is to set your GIT account. This GIT account is a local account used when you are committing local changes and pushing changes to your local repository folder.

Right click anywhere in your commit folder, hover over TortoiseGIT, and select “Settings” near the bottom of the context menu. In the left navigation page, click on “Git”. In the right navigation pane, put in any name and any email account (Figure 10). Since we are doing a local repository, the name and email account is not important for our purposes, therefore, you may ignore their importances for now.

From here, you can start placing your server data files inside your commit folder. For the purposes of this tutorial, I will be using a Minecraft server to serve as a placeholder for your dedicated server files.



(Figure 9): The hidden ".git" folder.



(Figure 10): The context menu and the Settings dialog, with "Git" selected.

After you have placed your server data files inside your commit folder, you should see a navy blue question mark icon overlay on all of your data files and folders (Figure 11). This icon overlay tells you that GIT has detected new files in your commit folder. We shall come back to this in a moment.

This next step involves a little bit of batch programming. For those who are not familiar with batch programming, you may refer to the following URL for more information:

http://www.robvanderwoude.com/batchfiles.php

I can assure you, it is not programming. Rather, it is just to change the GIT installation directory if you did not install GIT at the default installation location.

Now, run Notepad from your Start menu. In my case, I used Notepad++ for its syntax hightlighting features. You may choose to install Notepad++ from the following URL:

http://notepad-plus-plus.org/

The next step is to copy/paste the following code below into your preferred Notepad:

```
@echo off
REM ---- Locate git.exe from the GIT installation directory. ----
set GIT_PATH="C:/Program Files (x86)/Git/bin/git.exe"
%GIT_PATH% add -A
%GIT_PATH% commit -m "Auto-committed on %date%"
%GIT_PATH% push
```

If your default GIT installation directory is not the same as mine, you may need to change it accordingly. In my case, since I am using Windows 8 64-bit, my default installation directory is given in the code, shown above.

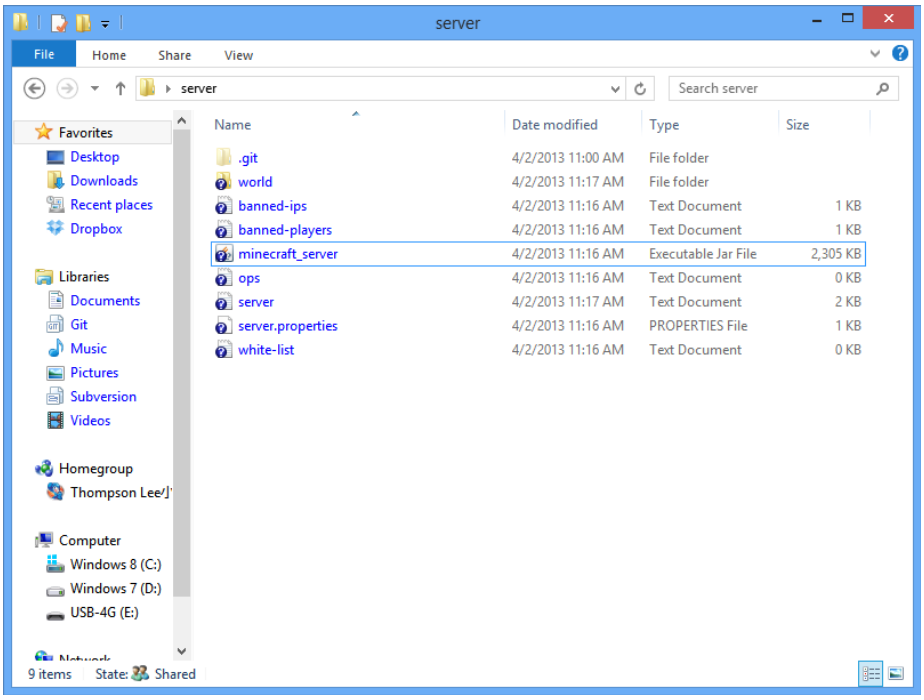
What the code does, is that it first calls on GIT to add all files that have been modified or changed in your commit folder. Then, it commits the added (or in this case, marked) files to the hidden folder, ".git", with the following message "Auto-committed on MMDDYY", where the MMDDYY stands for the date that this batch program script had run. Finally, it completes the backup by pushing the commit that was committed to the ".git" hidden folder to your repository folder.

Run the BATCH file, and it *should* automatically change the icon overlays to a green checkmark, telling you that all of your files and folders have been committed successfully to your repository folder (Figure 12).

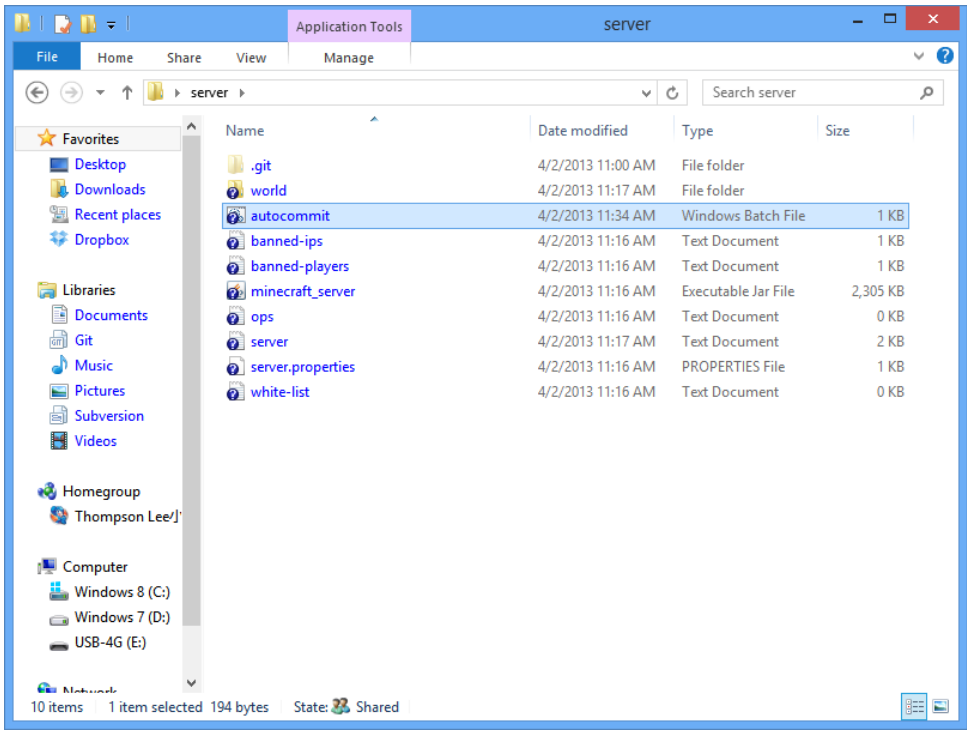
When I say "*should*", I meant that due to TortoiseGIT having the issue of not refreshing the icon overlays not completely fixed, you couldn't really tell if the files and folders have been committed and/or pushed to the repository. For more information, you may consult the following URL:

http://stackoverflow.com/questions/8137929/git-modified-sign-never-leaves

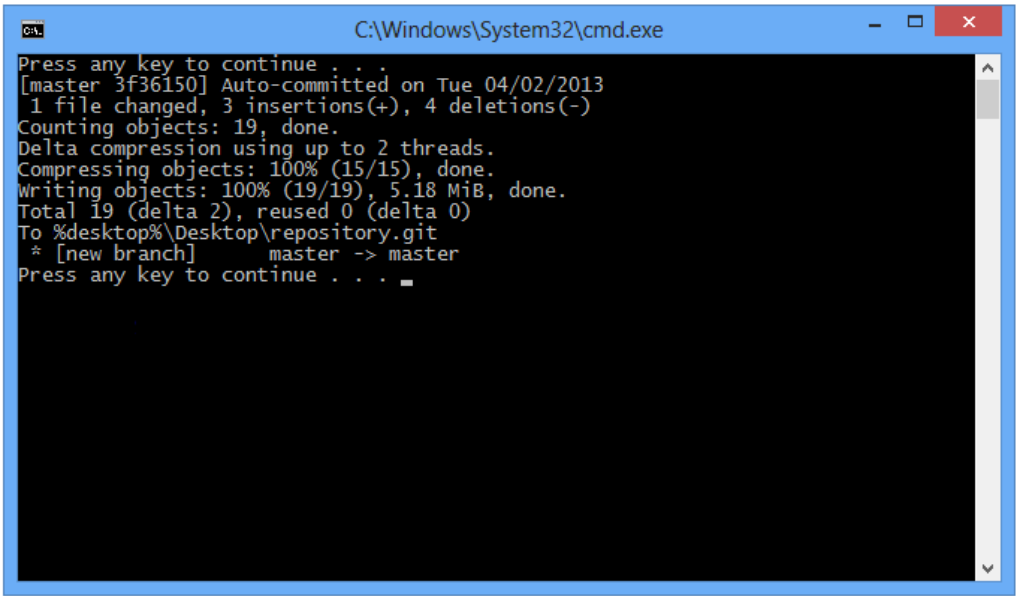
This ends the batch programming session.



(Figure 11): GIT detects new files.



(Figure 12a): The “autocommit.bat” file. Before execution.



(Figure 12b): The output of the GIT

Now, on towards automation.

In Windows 7, navigate to the Control Panel via Start Menu → Control Panel. In Windows 8, press the Start button, type "Control", and then press Enter. In the Control Panel, go to System and Security → Administrative Tools → Task Scheduler (Figure 13).

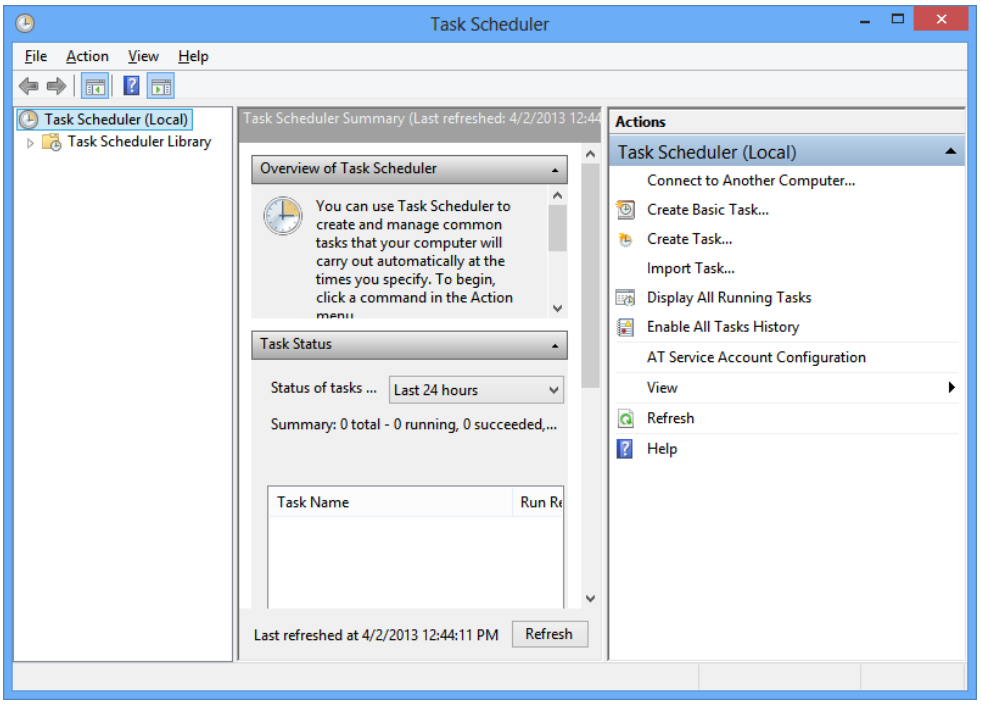
On the right pane, find and click on the option, "Create Task...". It is not to be confused with "Create Basic Task...", as it takes an additional step to setting it up. It should bring up the Task Wizard (Figure 14).

In the General tab, fill in the name and description of your task. In the Triggers tab, click on "New" to bring up a schedule wizard that aids you in setting up automation. Once this is set, in the Actions tab, press "New" to bring up a dialog asking you for the program/script you want to run. In this case, we want to run the BATCH file we have just copied/pasted/created in the commit folder. Browse to the BATCH file, select the file, and press OK. You may continue to set additional options in the Conditions and Settings tabs, if you liked. Once all of the options have been set, go ahead and click on OK to set the task in the Task Scheduler Library folder.

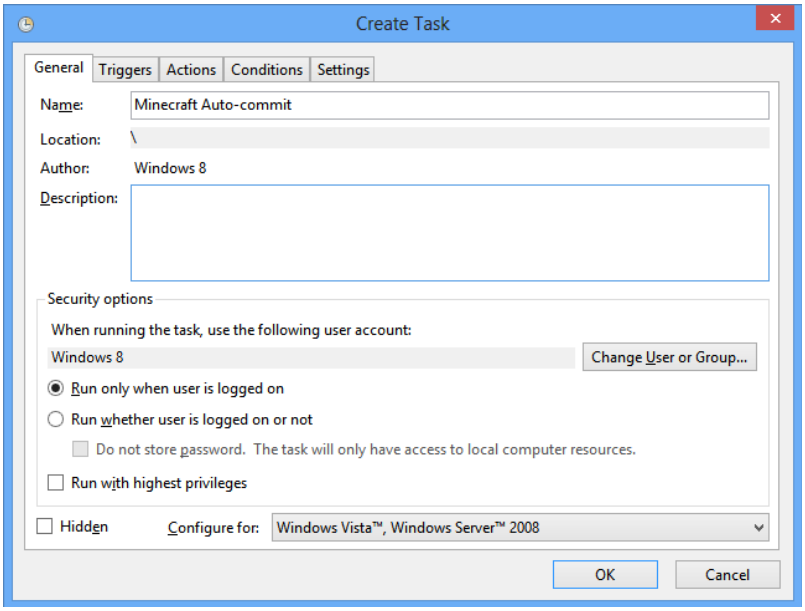
Once all of this is done, your task should be in the Task Scheduler Library folder, by click on the "Task Scheduler Library" folder on the left pane in Task Scheduler. From this point on, you will have to test and see if the BATCH file is working for GIT, if all modified files have been committed and pushed to the repository, check to see if your Task is running and working properly, and check to see if it causes substantial lag when it's committing and pushing while your dedicated server is running.

Other than all of those things, you have successfully created an automated backup system. Refer to the TortoiseGIT tutorials to learn how to revert your modified files to the last pushed commit version. If you wished to learn more about GIT commands, refer to the following URL:

http://gitref.org/



(Figure 13): Task Scheduler.



(Figure 14): Task Wizard.

There is one warning that should warrant a mention in this book. As in the case of substantial lagging, it usually occurs when any of the following conditions are met:

1. The auto backup system has been running for quite some time.
2. The total number of commits being pushed to the repository folder is in the thousands.

It would usually implies that either your auto backup system has been set to automatically commit

changes too frequently, or there aren't any big changes in your server files, and it is wasting resources. And those scenarios are very likely to happen when it comes to hosting a server.

If lagging were to occur while your dedicated server is being saved to your auto backup system, you may do the following simple checklist, just to be sure:

- If there are players in the server:
    - Ask if they experience any lagging.
    - Make sure that they know that your automatic backup system is running.
- If there are occurrences where other players are causing trouble:
    - Ask if other players are okay with rollbacks.
    - Make sure that when doing rollbacks, you do it when the server is offline.

This simple checklist is created out of experience I had when setting up an automatic backup system while hosting and maintaining a dedicated Minecraft server. You may have different experiences with your dedicated server. You may even have a different game. It all matters when it comes to making sure that the backup system is running smoothly and not causing any lag for other players.

Whatever you do, always periodically check if the system is running well, and that you know how to do a rollback in case something wrong happens. If you set the frequency of committing changes too low, chances are you might have to be wary of sudden changes in the server files. Otherwise, if you set the frequency too high, it will be hard to start a rollback when you found a fault.

Try to maintain a simple balance between the frequency of committing and the chances of rolling back the changes. The balance can be entirely different for different types of games and their respective server files.

Last, but not least, you have successfully learned how to create an automatic backup system, using GIT. Stay alert! Always have a backup plan.

Credits

Author: Thompson Lee
Illustrator: Thompson Lee

All images are screenshots from Windows 8 64-bit.
All URL links provided are for public consumption.

Published Date: April 3, 2013

Please give feedback whenever possible. It's always welcomed.