

# Git Crash Course

# Outline

- Introduction
- Getting Started
- Individual Versioning
- Central Collaboration
- Peer to Peer Collaboration
- Resources

# Download

- <http://git-scm.com>
- Main Documentation
- Source download for POSIX-compatible systems (Linux, Mac, etc.)



The screenshot shows the Git website homepage. At the top, the Git logo is followed by the tagline "everything-is-local". A search bar is in the top right corner. The main content area describes Git as a "free and open source distributed version control system" and highlights its "easy to learn" nature and "tiny footprint". A diagram on the right illustrates a branching model with stacks of code and colored arrows. Below this, a banner promotes "Try Git" in a browser. The middle section features four icons with text: "About" (advantages), "Documentation" (command reference, Pro Git book), "Downloads" (GUI clients, binary releases), and "Community" (mailing list, chat). To the right, a monitor displays the "Latest source Release 2.1.3" and a "Downloads for Mac" button. Below the monitor are links for "Mac GUIs", "Tarballs", "Windows Build", and "Source Code". At the bottom, a section titled "Companies & Projects Using Git" lists logos for Google, Facebook, Microsoft, Twitter, LinkedIn, Netflix, and others.

**git** —everything-is-local

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint** with **lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.

Learn Git in your browser for free with **Try Git**.

**About**  
The advantages of Git compared to other source control systems.

**Documentation**  
Command reference pages, Pro Git book content, videos and other material.

**Downloads**  
GUI clients and binary releases for all major platforms.

**Community**  
Get involved! Mailing list, chat, development and more.

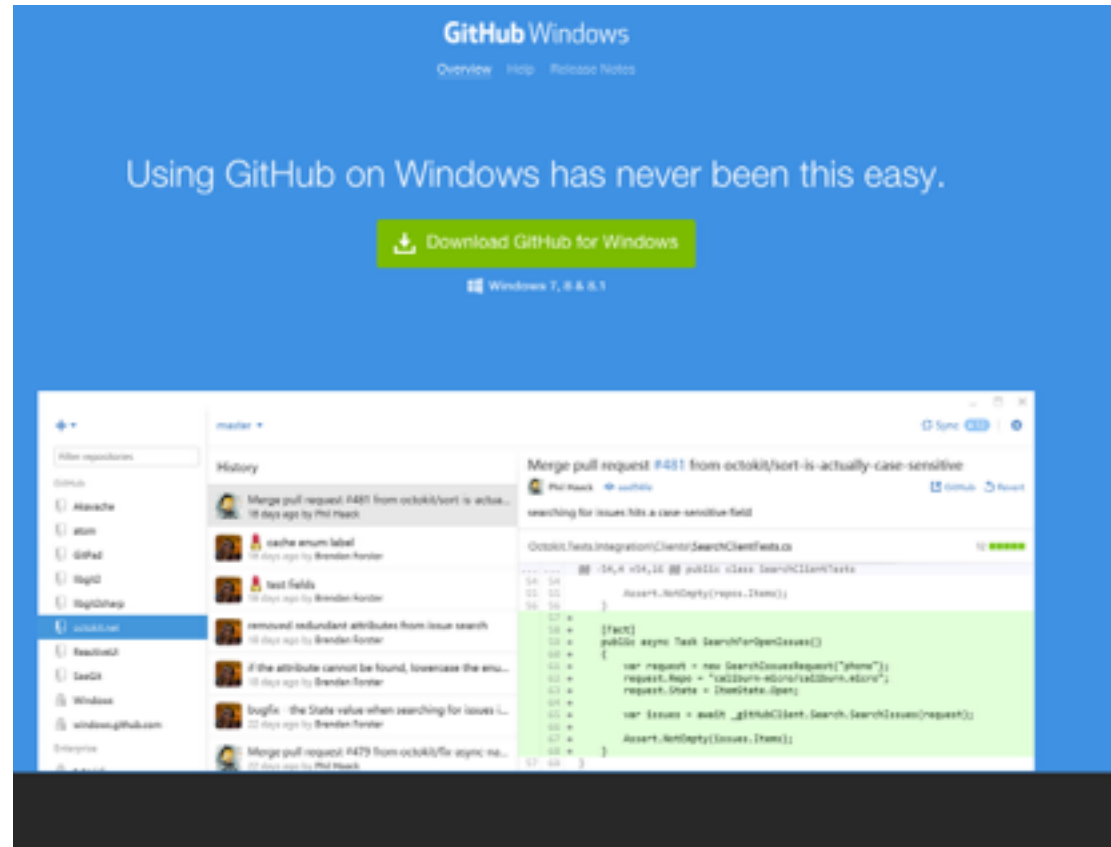
**Latest source Release**  
**2.1.3**  
Release Notes (2014-10-24)  
[Downloads for Mac](#)

[Mac GUIs](#) [Tarballs](#)  
[Windows Build](#) [Source Code](#)

**Companies & Projects Using Git**  
Google facebook Microsoft twitter LinkedIn **NETFLIX**  

# GitHub for Windows

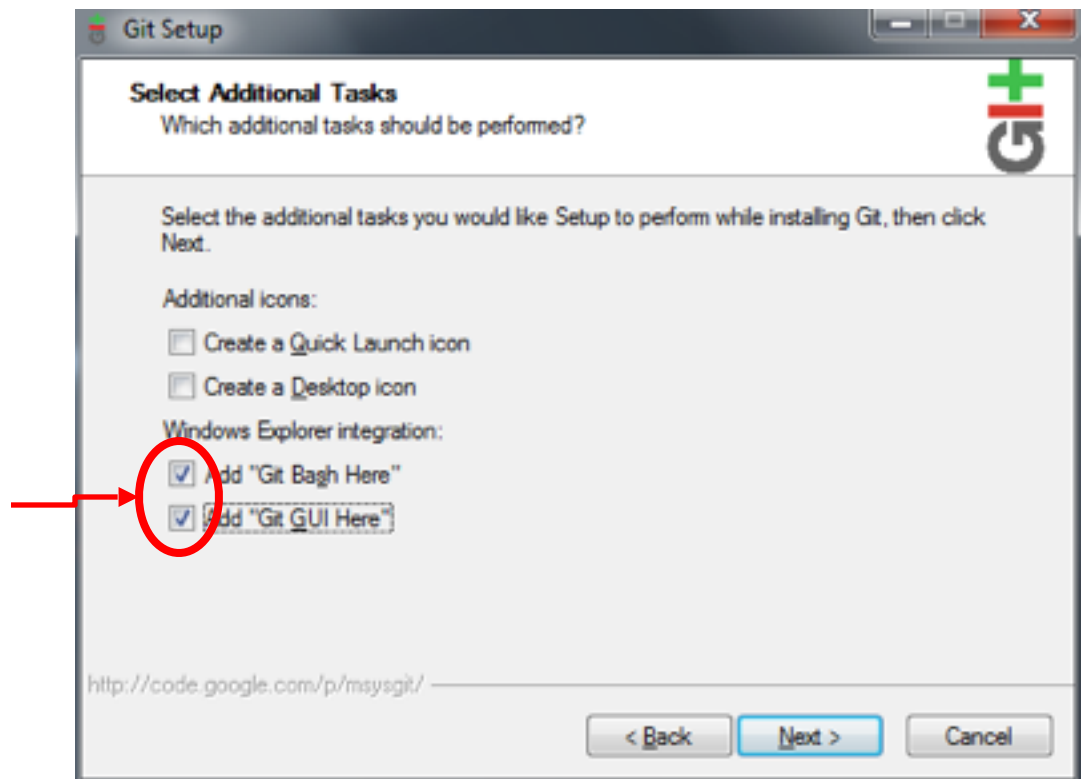
- <https://windows.github.com/>
- Alternate git(hub) client
- Vastly simplified interface
- I don't use it



# Installation

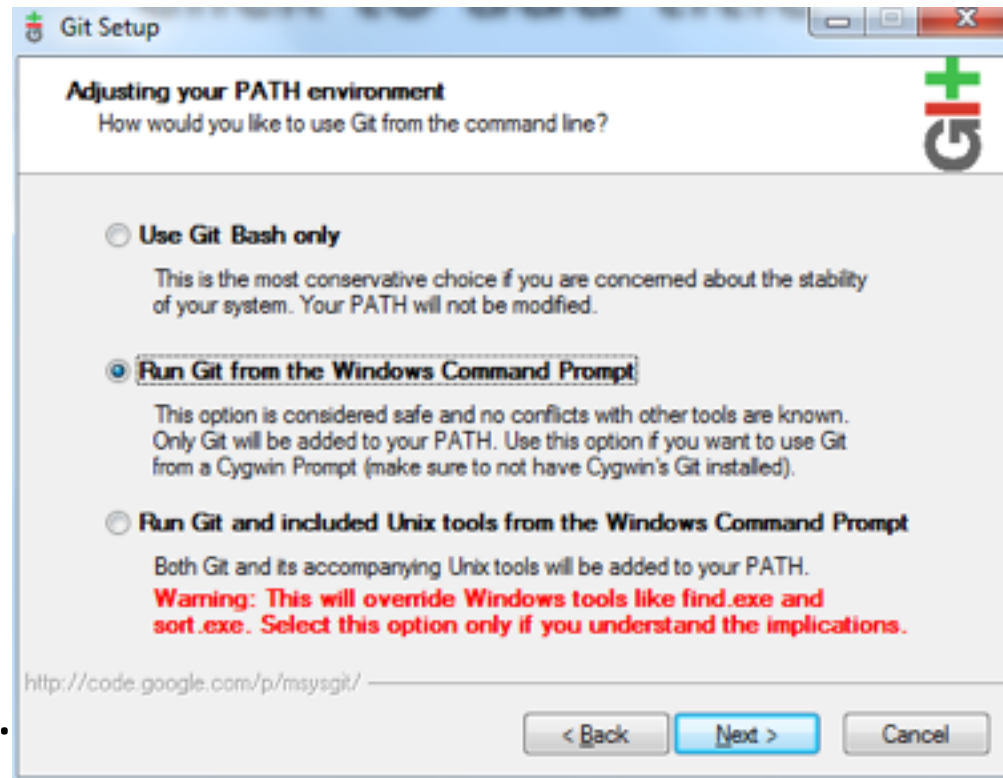
Launch the installer and follow the steps

- Creates shortcuts to the command prompt.
- It's usually a good idea to check one of this.



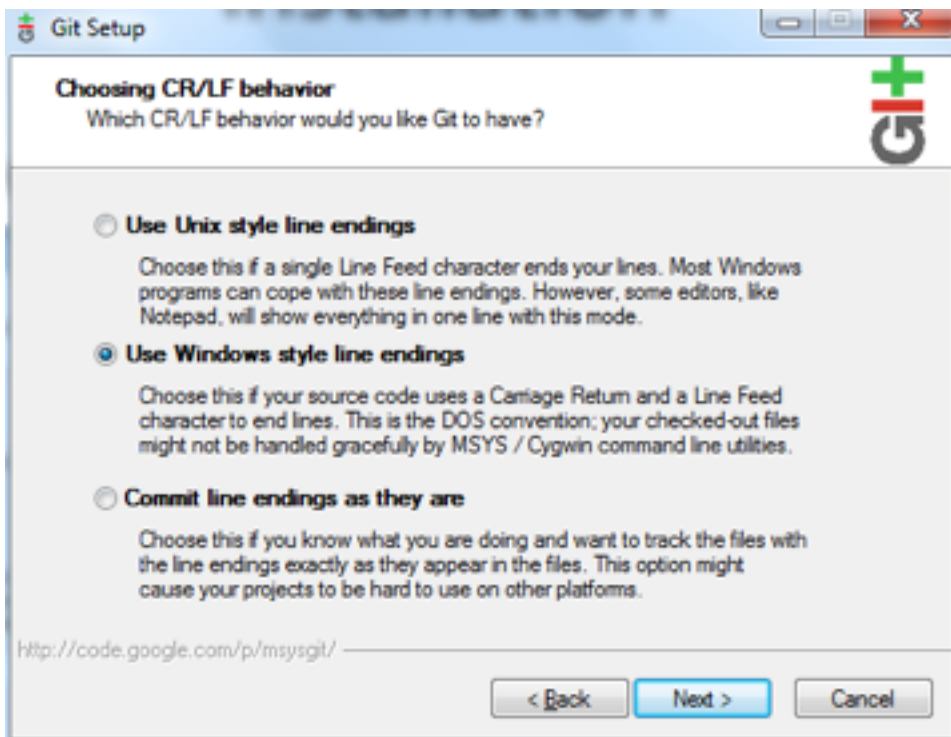
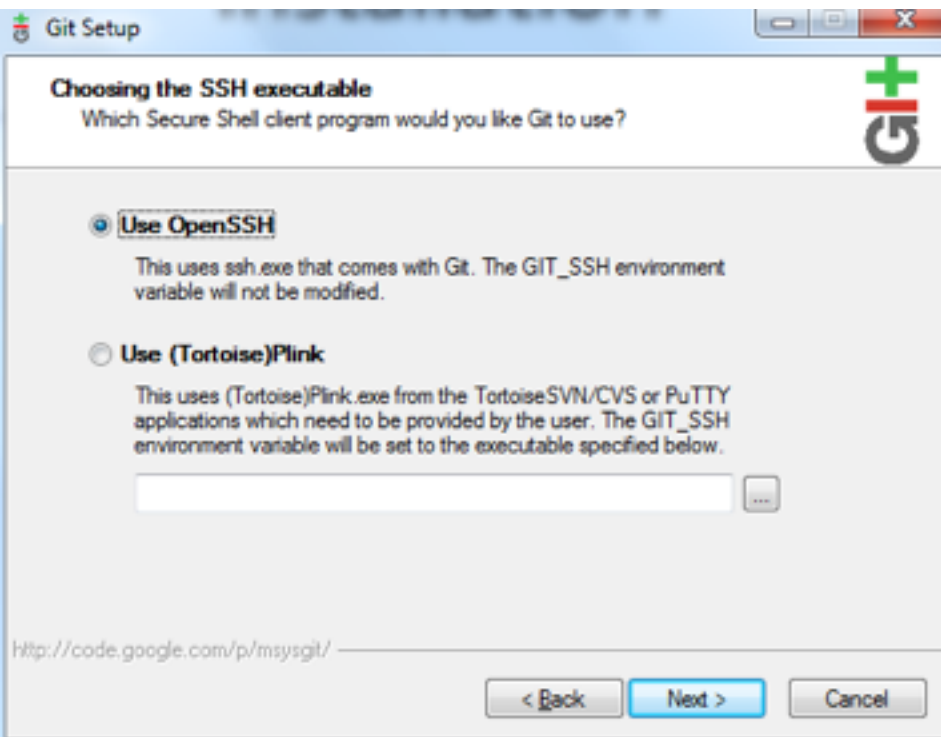
# Installation

- Use Git from
  - Command Prompt
  - Bash
    - The “command prompt” of Linux
    - Better support – you get colors!
    - Usually the better option.
- Select the 2nd option



# Installation

Accept the default settings for the next ones.



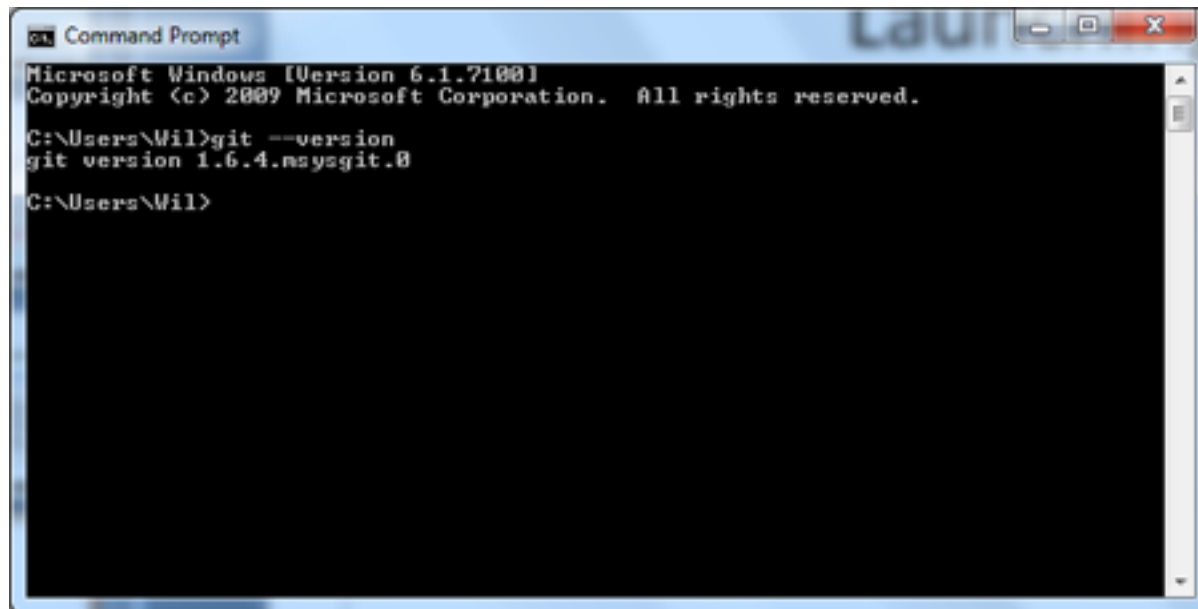
# Launching

- Command Prompt
- Git Bash
- Git GUI
- Explorer Extension



# Launching – Command Prompt

- Applies when you selected “Run Git from Command Prompt” during installation
- Launch cmd as you would normally



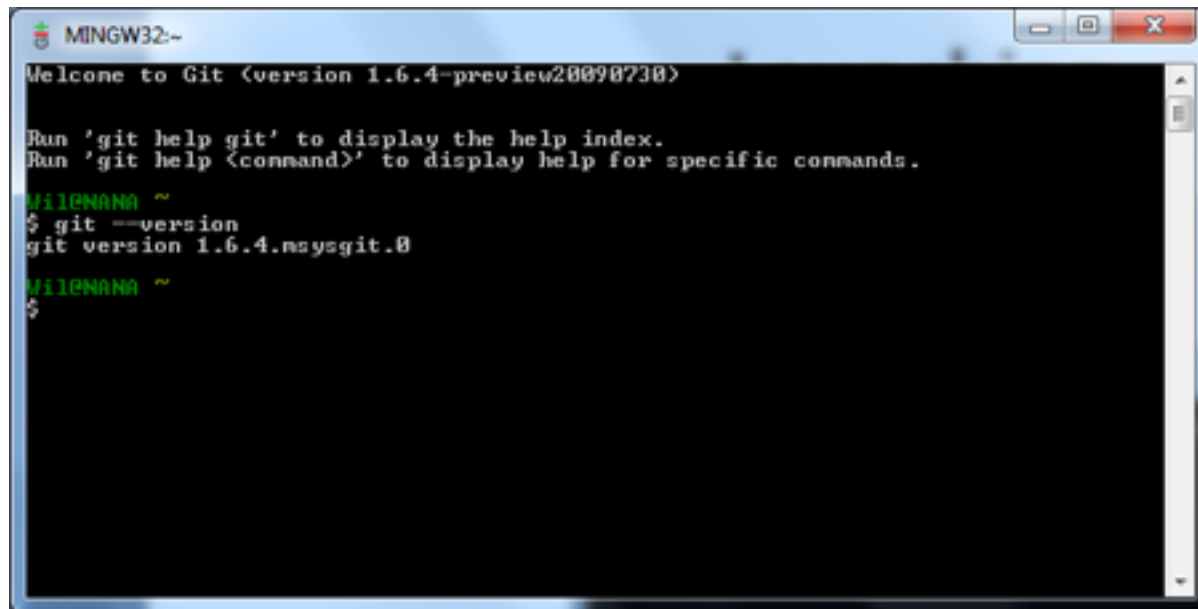
```
Command Prompt
Microsoft Windows [Version 6.1.7100]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Wil>git --version
git version 1.6.4.msysgit.0

C:\Users\Wil>
```

# Launching – Git Bash

- Linux Shell emulation – Linux tools available
- Launch by executing “Git Bash” from the start menu.



```
MINGW32~  
Welcome to Git (version 1.6.4-preview20090730)  
  
Run 'git help git' to display the help index.  
Run 'git help <command>' to display help for specific commands.  
  
Milewanna ~  
$ git --version  
git version 1.6.4.msysgit.0  
  
Milewanna ~  
$
```

# Launching – Git GUI

- GUI utility
- Makes some tasks easier
- You'll still need to learn how the entire system works though.



# Configuration

- Before starting to use Git, it would be helpful if some things are set up.
  1. Enable color output
  2. Set user info
  3. Generate SSH Key
  4. Set text editor
  5. Set merge tool

# Configuration – Enable Color

- Type the following in Git Bash or CMD

```
git config --global color.diff auto  
git config --global color.status auto  
git config --global color.branch auto
```

- Note: This will only be effective for Bash sessions.

`git config` is one of the git commands that does manipulates configurations

# Configuration – Set user info

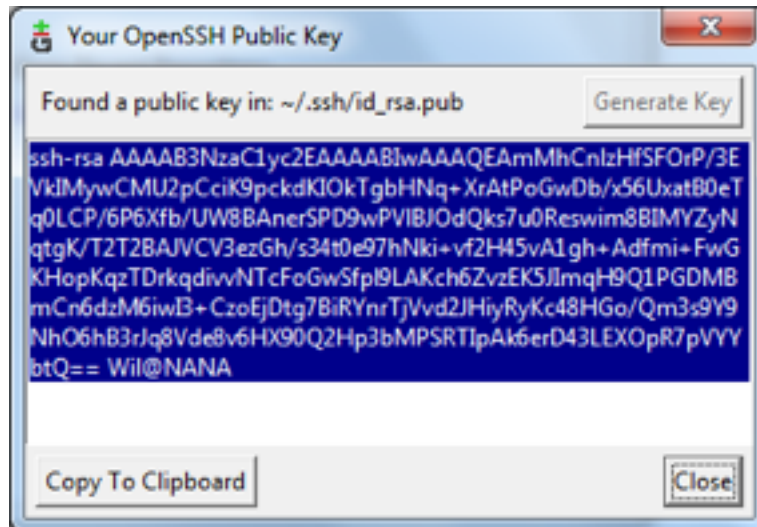
- Type the following in Bash or CMD

```
git config --global user.name "Your Name Comes Here"  
git config --global user.email you@yourdomain.example.com
```

- Of course, use your own name and email.
- Like your username and email in Facebook, this identifies you when you start collaborating with others.

# Configuration – SSH Key

- Open Git GUI
- Go to Help > Show SSH Key
- If you don't have an SSH Key yet, press “Generate Key”

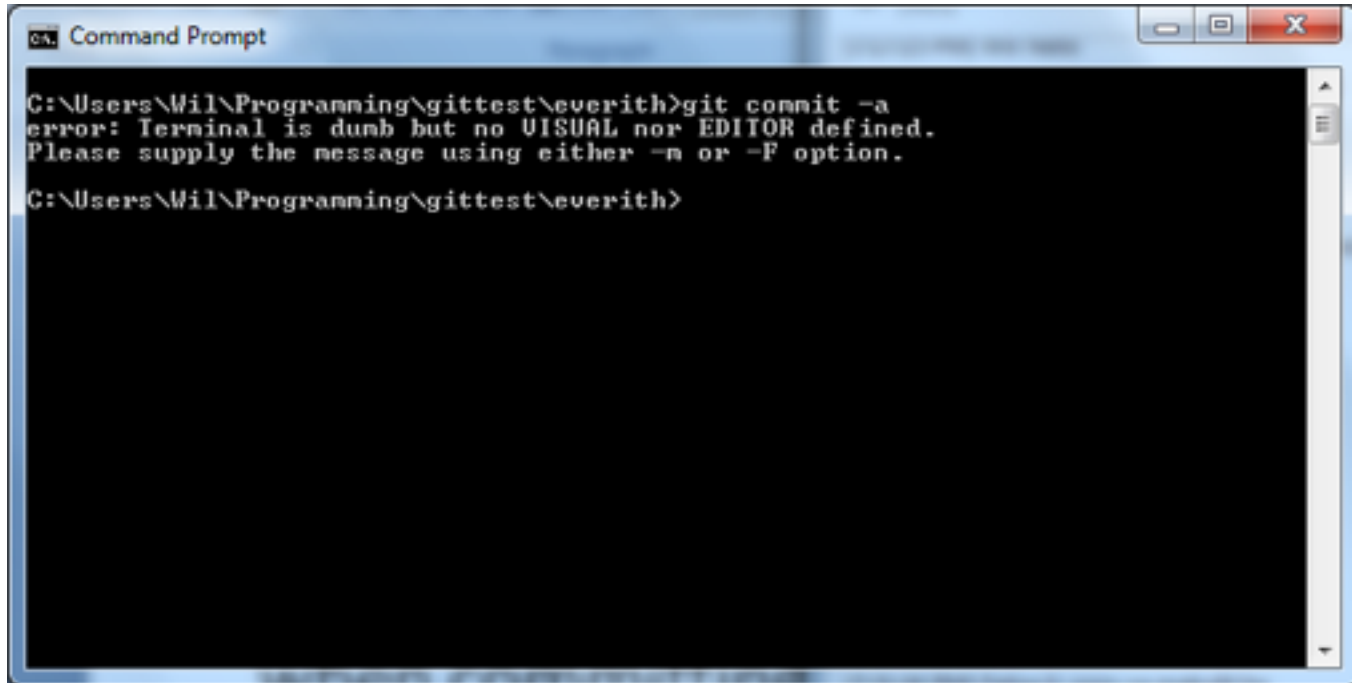


# Configuration – Text Editor

- Every commit in Git needs a message.
- You can explicitly pass it in the command when committing or have git automatically launch a text editor for you.
- The default is vim for Bash but it might not be the best for everyone.
- You might want to set your own text editor for it.



# Configuration – Text Editor



```
Command Prompt
C:\Users\Wil\Programing\gittest\everith>git commit -a
error: Terminal is dumb but no VISUAL nor EDITOR defined.
Please supply the message using either -m or -F option.
C:\Users\Wil\Programing\gittest\everith>
```

# Outline

- Introduction
- Getting Started
- Individual Versioning
- Central Collaboration
- Peer to Peer Collaboration
- Resources

# Individual Versioning

- Initializing a Repository
- Adding Files
- Committing
- Tagging
- Logs
- Botched Commits
- Rolling Back
- Branching
- Merging

# Initializing a Repository

- Git works on “blessed” directories called repositories.
- To bless a directory into a repository

# Central Server

- Cloning an existing repository
- Remote branches
- Getting updates

# Cloning an existing Repo

- You can work with a centralized server by cloning, working and then pushing changes to the server.

```
git clone <url>  
git clone http://www.someserver.com/repo/proj\_a  
git clone user@server.com:repo/proj_a
```

# Pulling from the server

- You can pull in the new stuff from the repository by using git pull.

```
git pull
```

- No need to specify the URL since git knows where you cloned it from.

# Committing and Pushing

- When you're ready, add your files, commit and push to the server.

```
git add .  
git commit -m"Commit message here"  
git push
```

- Use “git status” to show changed files
- Leave a sensible commit message



# Conflicts and Merging

- Editing the same file may generate conflicts that git can't resolve on its own
- Files with conflicts during a merge/pull/fetch will have something like the following (next slide)

# Conflicts and Merging

Here are lines that are either unchanged from the common ancestor, or cleanly resolved because only one side changed.

```
<<<<<< yours:sample.txt
```

```
Conflict resolution is hard;
```

```
let's go shopping.
```

```
=====
```

```
Git makes conflict resolution easy.
```

```
>>>>>> theirs:sample.txt
```

```
And here is another line that is cleanly resolved or unmodified.
```

# Conflicts and Merging

- Markers are <<<<<<, ===== and >>>>>>
- Your changes are normally between < and =
- Their changes are normally between = and >

# Conflicts and Merging

- Edit the file by removing the markers and changing the relevant portions
- When done, commit the result and push if necessary.

# Remote Branches

- Remote branches are branches that can be found in another repository other than your own
- You should never EVER deal with remote branches directly
- Disregarding the line above will have severe consequences to your sanity (and whoever owns the remote branch)

# Remote Branches

- Instead, you create a local branch to keep track of the remote branch.
- Think of it as mirroring
- Use the following command to track remote branches

```
git checkout --track -b <branch> origin/branch  
git checkout --track -b local_experiment origin/experiment
```

# Remote Branches

- You can switch to the newly created branch and do changes there like a normal branch
- Changes to and from the remote branch will be reflected by the local branch and vice versa when a push or pull command is executed.

# Tips: Central Model

- Pull from the repository before committing. That way you can resolve conflicts before pushing them (and troubling others)
- Give short sensible commit messages so everyone knows what you changed or did.



# Peer-to-Peer

- “Send” workflow
  - Generating patches
  - Generating bundles
- “Pull” workflow
  - Serving your git repository
  - Bare repositories

# Send: Generating Bundles

- You can also send a portion of the repository history through bundles
- Git bundle command

```
git bundle create <file> <git-rev-list args>  
git bundle create my_bundle.git master  
git bundle create my_bundle2.git HEAD^2..HEAD  
git bundle create my_bundle3.git -10 master  
git bundle create bundle4.git --since=8.days master
```

# Send: Generating Bundles

- Send the bundle to someone using e-mail or usb
- Recipient can then pull from the bundle like a normal repository.

```
git clone <path to bundle> <folder_name>  
git clone C:\my_bundle1.git my_project  
git clone /home/user/my_bundle2.git my_project
```

# Send: Generating Bundles

- You can make an updated bundle, send it and then the other person can pull from it by replacing the original bundle file.
- After replacing the bundle file, simply do a “git pull” from the project directory