Additional Resources

If you aren't a version control expert after reading this booklet, don't worry. You have only been introduced to the laundry list of features provided by Git and GitHub. What you have learned should be enough to make use of core version control principals but there is plenty more to explore. Try reading some of the resources below or just trying new things by yourself with an experimental repository.

To learn more about version control, Git, or GitHub, check out the following websites:

- https://try.github.io/
- http://git-scm.com/book/
- https://help.github.com/

To learn more about command line interfaces in Windows, try these websites:

- http://codeproject.com/Articles/457305/Basic-Git-Command-Line-Reference-for-Windows-Users
- http://gitref.org/
- https://dosprompt.info/

Graphical interfaces for using Git are available from the following websites:

- https://windows.github.com/
- http://sourcetreeapp.com/

Using Git & GitHub

Page Contents

Common Uses for Git

When working as a team on a project with many documents, it can be very frustrating trying to keep things organized. When passing files around, it can be a struggle to know which version is the most recent. Two team members may want to do work on the same file at the same time for entirely different reasons. If they each work on their copy of the file, which do you keep at the end? How do you combine their changes?

Although Git is most often used for programming projects, it works well with any projects which use plain-text files. This includes web development, LaTeX, markup, or simple text documents.

A file server or shared folder can help keep all of the files in one place, but editing with multiple people is still troublesome. For some files, a collaborative editor such as Google Docs may help mitigate the issue, but simultaneous editing can be confusing and there are few editors which are truly designed for it. What we really need is a tool to allow multiple people to work on the same file and easily combine their changes.

Primary Features of Git

- Can combine multiple sets of changes to a single text file
- Shows which contributor is responsible for each change
- Allows changes to be removed or reverted
- Makes it easy to finalize and distribute specific versions

Git is a **version control** software solution. It is designed to allow multiple contributors to edit a pool of files located in a **repository**. GitHub is a website which hosts Git repositories. Although other version control solutions and repository hosting sites exist, Git and GitHub are particularly popular and powerful.

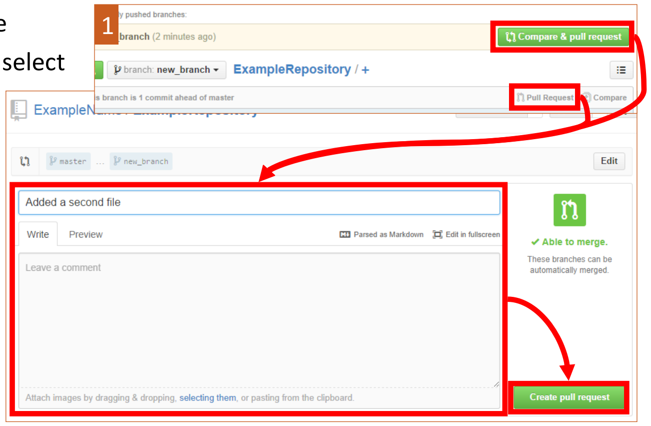
Expectations and Assumptions

This booklet is designed to teach fundamental version control concepts and explain how to use these concepts with Git and GitHub. Both tools offer a plethora of additional features, but after finishing these instructions you will be able to set them both up and properly use their primary features on a Windows computer. Since Git is most often used via a command line interface, basic instructions on how to use the Git Bash software will also be provided.

Before beginning these instructions, it is assumed that you are familiar with some basic functionality of a modern Windows computer. This includes the ability to start programs, navigate the file system using the Windows Explorer software, and navigate to and interact with web pages. If any of these tasks sound unfamiliar, it is recommended that you learn them before continuing with this booklet.

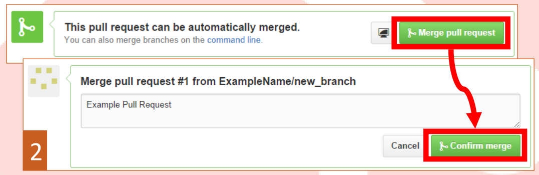
Using Pull Requests

Often, a new branch is used in the workspace to make a series of changes. After these changes have been committed, reviewed, and pushed to the repository they need to be merged into the master branch. **Pull requests** provide an interface for reviewing and merging changes from one branch to another.



1 **CREATE REQUEST:** On the GitHub repository page, select the branch you want to merge and **click on the "Pull Request" link** located above the file list. Alternatively, if the branch is new GitHub will show an orange box with a "Compare & pull request" button. GitHub will provide a simple form to create the pull request with a title and description. Above the form is a blue box which you can use to specify which branch to merge into. It should default to master, which is what we want. Below the form is a list of commits and a "diff," or comparison of changes between the two branches. Once you know that the changes are correct, **fill out the form and click the "Create pull request" button**.

2 **MERGE:** After creating the pull request, you will be brought to the web page for that request. Normally this would provide an opportunity for other contributors to inspect your changes and provide feedback. When the pull request has been reviewed, merge it by **clicking on the "Merge pull request" button, then the "Confirm merge" button**. A message can be provided if desired.


This pull request can be automatically merged.
You can also merge branches on the command line.
Merge pull request
Merge pull request #1 from ExampleName/new_branch
Example Pull Request
2
Cancel
Confirm merge

After merging a pull request, the branch may not be needed any more. The pull request page will offer a button to delete the branch on the repository. To delete the branch on your workspace, use the command `git branch -D <branch>` where <branch> is the name of the branch you wish to delete.

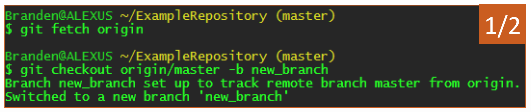
Return to the repository page by clicking on the name of the repository at the top of the pull request page. The master branch on the repository change should now look the same as the branch which was merged.

Using Branches

Branches were introduced previously as a useful means for contextualizing a series of commits. They allow for multiple versions of the repository to exist based on the "master" commit path. To create a new branch, you must specify a commit within an existing branch to act as a starting point. Generally, you start a new branch from the last commit in another branch (usually the "master" branch).

1 FETCH: First, we need to make sure that the workspace is completely aware of any new commits on the repository. Since it does not automatically update whenever new commits or branches are added to the repository, we must manually tell the workspace to update. **Type the command `git fetch origin`** to update local information about the origin remote.

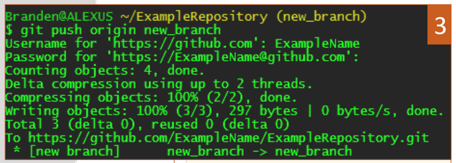
2 CREATING A BRANCH: To create a new branch from master, **enter the command `git checkout origin/master -b new_branch`**. Again, we see "origin" and "master" in the command which can be substituted for any remote or branch name on the specified remote.


1/2

```
Branden@ALEXUS ~/ExampleRepository (master)
$ git fetch origin

Branden@ALEXUS ~/ExampleRepository (master)
$ git checkout origin/master -b new_branch
Branch new_branch set up to track remote branch master from origin.
Switched to a new branch 'new_branch'
```

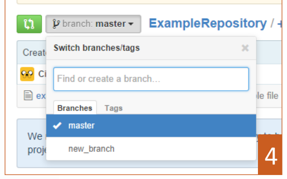
The "-b new_branch" section tells Git to create a completely new branch based on origin/master with the name new_branch. Notice the branch name change from "master" to "new_branch". To switch back to master or another branch, use the command `git checkout <branch>` where <branch> is the name of the desired branch. Just typing the command `git branch` will also show you a list of all branches on your workspace.

3 COMMIT: Once we have a new branch, we can change something and commit normally (page 11). Almost all of the steps are the same as when committing to master. The only difference is that instead of pushing with `git push origin master`, we use the command `git push origin new_branch`. **Create a new file, add the change, commit, and push new_branch to the repository**.


3

```
Branden@ALEXUS ~/ExampleRepository (new_branch)
$ git push origin new_branch
Username for 'https://github.com': ExampleName
Password for 'https://ExampleName@github.com':
Counting objects: 4, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 297 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/ExampleName/ExampleRepository.git
 * [new branch]      new_branch -> new_branch
```

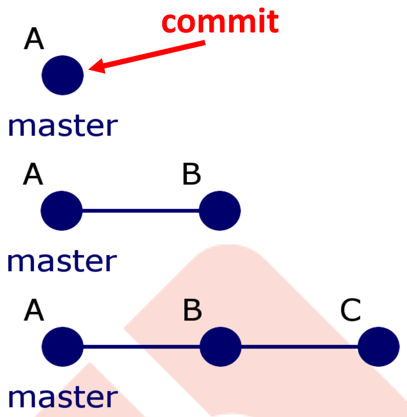
4 VIEWING REPOSITORY: **Navigate to the repository on GitHub**. When you open the repository page, you can see any changes you made on the master branch. If you want to see changes for new_branch, you must select it from the branch dropdown list. **Click on the branch selection box and select new_branch**. Now the page should show all changes for the newly created branch.



Introduction to Version Control

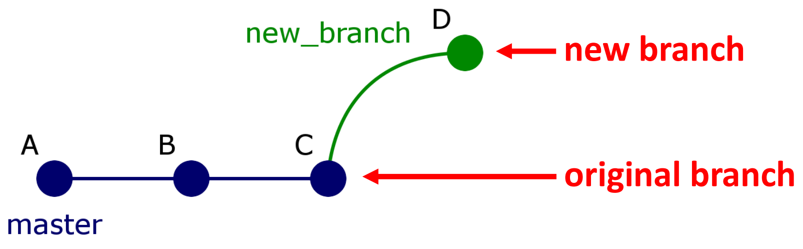
Instead of saving a whole file, version control software works by saving sets of changes made to the files within repository. These groups of changes are called **commits**. By using commits, users are able to work on the same file simultaneously because their commits will remain separate. Sometimes two commits will conflict with each other when both are applied, but in these cases the version control software can identify which changes are affected so that they can be easily resolved.

When a repository is first created, it will be empty. To make changes to the repository, a user will need to set up a local **workspace**. The workspace is a folder which contains local copies of the files from the repository. It allows you to **pull** down commits from the repository - updating all of the files in the workspace - and **push** new commits up to the repository. After the workspace is ready, you will be able to push your first commit.



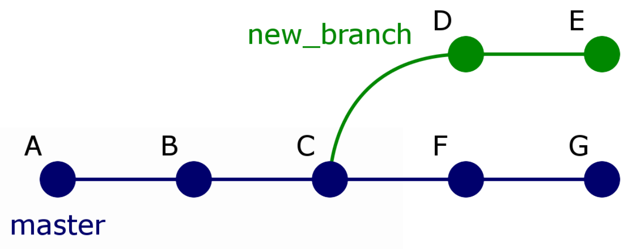
In the commit diagram to the right, the first commit is represented by the dot labeled "A." This commit is the first set of changes made to the repository, so it will involve creating one or more files. Each commit after A is attached to the previous commit, creating a chain or series (see commits "B" and "C" in the diagram). These commits can involve adding new files, changing existing ones, or even deleting files.

As a general rule, the changes within a commit should be related. This is because commits can be removed if it is later decided that the changes are not wanted. If many unrelated changes are a part of the same commit, it is likely that only some of the changes will need to be removed but the whole commit will have to be removed anyway. The unrelated changes will then have to be added back to the repository manually with a separate commit. Keeping changes related also helps a team understand the purpose behind a commit without inspecting each individual change.

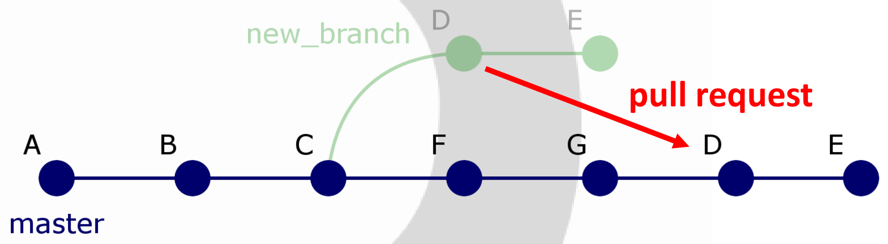


In addition to commits, Git offers features called **branches**. A branch is simply a series of commits, which result in a particular set of files. By default, the repository is created with a single branch named "master." This is usually the branch with "official" files that everyone is working on.

Like a tree, branches split from other branches. In the top diagram a new branch, appropriately named "new_branch," is created based on commit C of the master branch. The files in master will still only consist of the changes in commits A, B, and C. However, the files in new_branch will consist of the changes in commits A, B, C, and D. We can continue adding additional commits to either branch and they will remain separate paths.

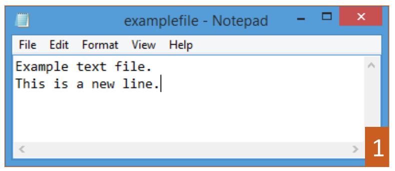


Often times branches are used when making a set of changes on the workspace. A new branch is created on the workspace based on the latest commit in the master branch. Commits are added to the branch until all desired changes have been made, then the branch is pushed up to the repository. In the repository, a feature called a **pull request** allows the commits in a branch to be reviewed by the other contributors and then merged into the master branch. In the example diagram, a pull request from new_branch into master would append the commits D and E onto the end of master, immediately after commit G.



The repository also provides **forks**. A fork is basically a copy of a repository for an individual contributor. A contributor can push to his fork instead to avoid adding branches to the primary repository. Pull requests can then be made from branches on a fork to branches in the primary repository.

Stashing and Reverting Changes

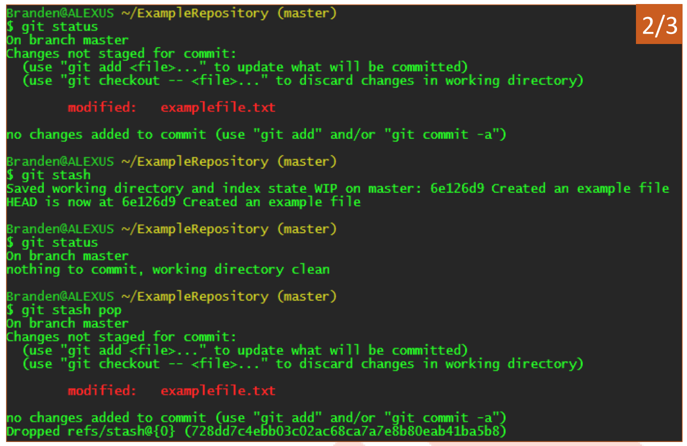


```
Example text file.
This is a new line.
```

Sometimes you might be in the middle of making changes for something when a higher priority change comes up. How do you switch to making a separate set of changes without committing unfinished work or loosing your progress? Git provides us with the `git stash` command for these situations.

1 **CHANGE SOMETHING:** **Make a simple modification to a file in the repository and save the change**. The `git status` command should reflect this change.

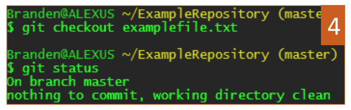
2 **STASH:** Next, **enter the command `git stash`** to temporarily revert the file back to what is on the repository. The changes are placed at the top of a "stack" of changes. If you do not immediately notice the changes, you may need to re-open the file. Once again, `git status` will confirm the changes by showing you that there are no modified files. Now your workspace is ready to work on another task.



```
Branden@ALEXUS ~/ExampleRepository (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   examplefile.txt

no changes added to commit (use "git add" and/or "git commit -a")

Branden@ALEXUS ~/ExampleRepository (master)
$ git stash
Saved working directory and index state WIP on master: 6e126d9 Created an example file
HEAD is now at 6e126d9 Created an example file

Branden@ALEXUS ~/ExampleRepository (master)
$ git status
On branch master
nothing to commit, working directory clean

Branden@ALEXUS ~/ExampleRepository (master)
$ git stash pop
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   examplefile.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (728dd7c4ebb03c02ac68ca7a7e8b80eab41ba5b8)
```

3 **POP:** After the task has been finished and committed, you can **recover your previous changes with the command `git stash pop`**. This command "pops" the changes from the top of the stack of changes which, as long as no other changes have been stashed, will be the changes you made previously.

Sometimes you modify a file and later decide you want to remove those modifications. You could try to undo all the work manually, but there is an easier way. The `git checkout` command is used to copy content from the repository to your local workspace. It can be used to copy entire branches or just single files or folders. In this case, we only want to checkout the file we modified earlier.

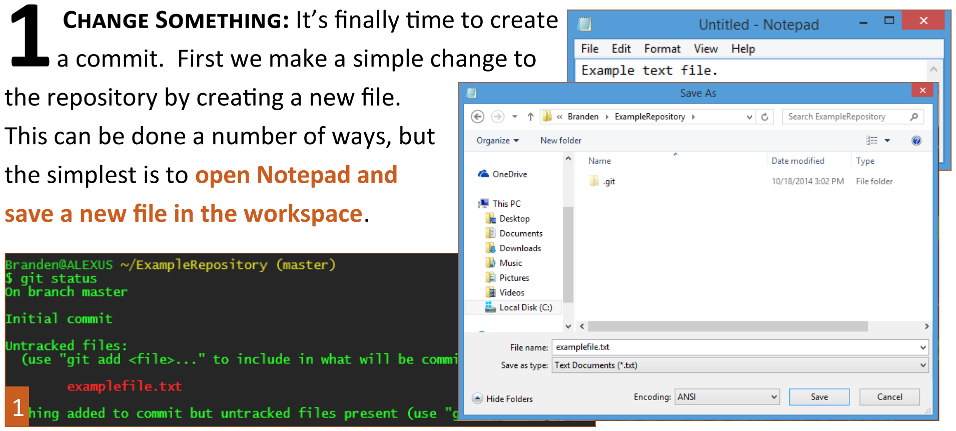


```
Branden@ALEXUS ~/ExampleRepository (maste
$ git checkout examplefile.txt

Branden@ALEXUS ~/ExampleRepository (master)
$ git status
On branch master
nothing to commit, working directory clean
```

4 **REVERT:** To checkout the file, **enter the command `git checkout <path>`** where <path> is the relative path of the designated file. Again, the `git status` command should reflect the changes by no longer showing the file in the change list.

Committing Changes

1 Change Something: It's finally time to create a commit. First we make a simple change to the repository by creating a new file. This can be done a number of ways, but the simplest is to **open Notepad and save a new file in the workspace**.


Untitled - Notepad

Example text file.

Save As

Name: .git — Date modified: 10/18/2014 3:02 PM — Type: File folder

File name: examplefile.txt

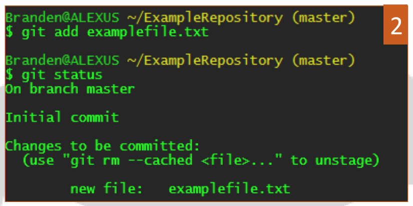
Save as type: Text Documents (*.txt)

Encoding: ANSI

```
Branden@ALEXUS ~/ExampleRepository (master)
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be commi

        examplefile.txt

hing added to commit but untracked files present (use "g
```

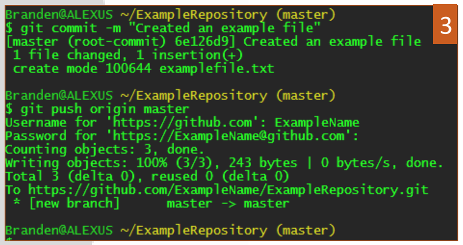
While in the workspace on Git Bash, try entering the command `git status`. This shows us that a file has been created or modified but is not selected to be a part of the commit.

2 Add Change: To select files to be committed, **use the command `git add <path>`** where <path> is the relative path of the designated file or folder. Also, the command `git add -A` can be used to add all files in the list. Now if you enter the `git status` command again, you can see that the file had been "staged" for the commit.


```
Branden@ALEXUS ~/ExampleRepository (master)
$ git add examplefile.txt

Branden@ALEXUS ~/ExampleRepository (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   examplefile.txt
```

A file can be removed from the staging list by typing `git reset <path>` where <path> is the path of the designated file. The `git reset` command can also unstage all files in the commit if <path> is omitted entirely.

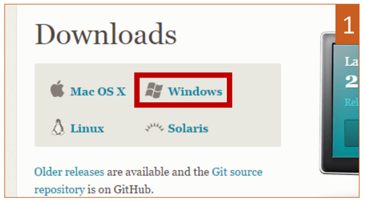
3 Commit: To finalize the commit, **enter the command `git commit -m "<Message>"`** where <Message> is a brief description of what has been changed. The message part may seem trivial, but it is important to communicate why a change was important to others and possibly even your future self.


```
Branden@ALEXUS ~/ExampleRepository (master)
$ git commit -m "Created an example file"
[master (root-commit) 6e126d9] Created an example file
 1 file changed, 1 insertion(+)
 create mode 100644 examplefile.txt

Branden@ALEXUS ~/ExampleRepository (master)
$ git push origin master
Username for 'https://github.com': ExampleName
Password for 'https://ExampleName@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 243 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/ExampleName/ExampleRepository.git
 * [new branch]      master -> master

Branden@ALEXUS ~/ExampleRepository (master)
```

4 Push: Lastly, to add the commit to the repository **use the command `git push origin master`**. "Origin" is the name of the remote which references the primary repository and "master" is the name of the current branch. You will need to provide your GitHub name and password to push.

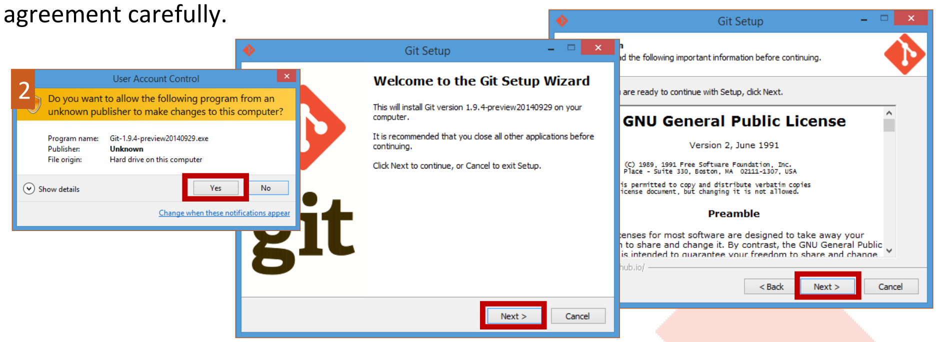
Installing Git on Windows

1 Download: To get the installer, **navigate to** http://git-scm.com/downloads on your preferred browser. Once there, **select the "Windows" link** from the box under the "Downloads" header. If you are using another operating system, select the appropriate link. Although the installer will not be exactly the same, the installation steps should be very similar to those for Windows. After clicking the link, the file should automatically download.


Downloads

Older releases are available and the Git source repository is on GitHub.

2 Start Installer: Locate the file and **run the installer** by opening it. If a popup asks if you want to allow the installer to execute, select "Yes." The installer should greet you with a welcome page. **Click the "Next" button twice** to skip both the welcome page and the license agreement. Don't forget to "read" the license agreement carefully.


User Account Control

Do you want to allow the following program from an unknown publisher to make changes to this computer?

Program name: Git-1.9.4-preview20140929.exe

Publisher: Unknown

File origin: Hard drive on this computer

Git Setup

Welcome to the Git Setup Wizard

This will install Git version 1.9.4-preview20140929 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Git Setup

d the following important information before continuing.

are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

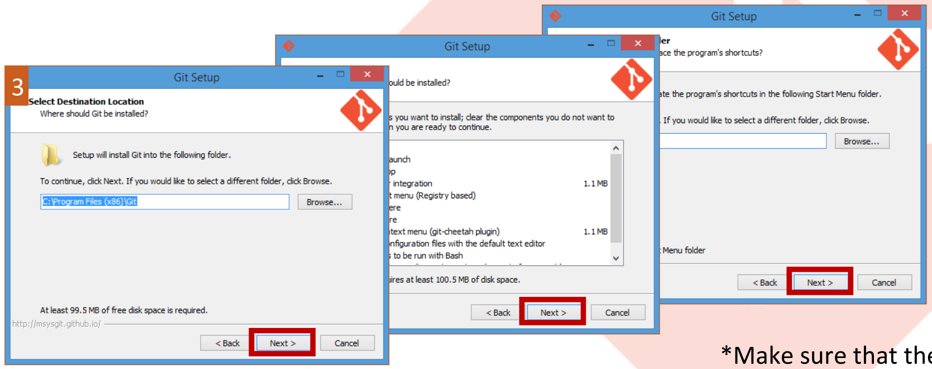
(C) 1989, 1991 Free Software Foundation, Inc.
Place - Suite 330, Boston, MA 02111-1307, USA

is permitted to copy and distribute verbatim copies
icense document, but changing it is not allowed.

Preamble

censes for most software are designed to take away your
to share and change it. By contrast, the GNU General Public
is intended to guarantee your freedom to share and change

3 Configuration: The next three pages ask about where to install the software, which components should be included, and where the start menu shortcuts should be placed. Only modify the values on these pages if you feel comfortable with them. For most users, the default parameters are appropriate for all three pages. **Click the "Next" button three more times** to continue.


Git Setup

Select Destination Location

Where should Git be installed?

Setup will install Git into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files (x86)\Git

At least 99.5 MB of free disk space is required.

Git Setup

uld be installed?

s you want to install; clear the components you do not want to n you are ready to continue.

aunch
p
integration 1.1 MB
t menu (Registry based)
ere
re
text menu (git-cheetah plugin) 1.1 MB
nfiguration files with the default text editor
 to be run with Bash

res at least 100.5 MB of disk space.

Git Setup

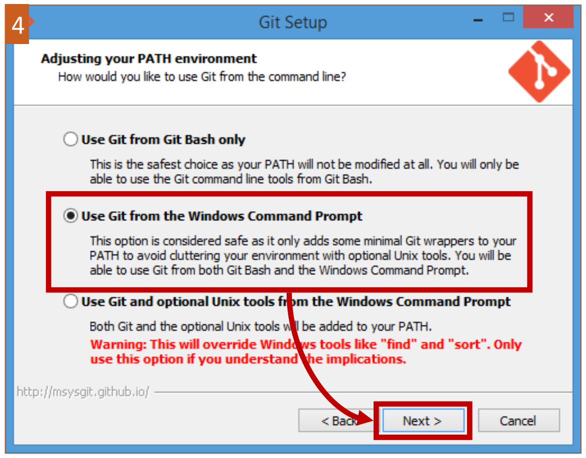
ce the program's shortcuts?

ate the program's shortcuts in the following Start Menu folder.

If you would like to select a different folder, click Browse.

Menu folder

*Make sure that the



Adjusting your PATH environment
How would you like to use Git from the command line?

( ) **Use Git from Git Bash only**
This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

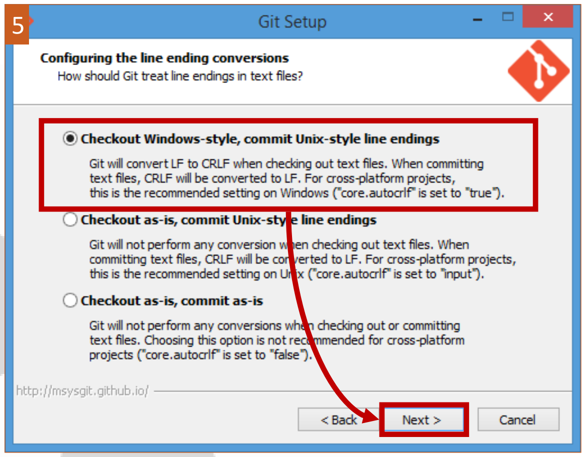
(●) **Use Git from the Windows Command Prompt**
This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

( ) **Use Git and optional Unix tools from the Windows Command Prompt**
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

http://msysgit.github.io/

4 Command Interface: The next page in the Git installer provides you with three options. The first option, "Use Git from Git Bash only" is the default selection. However, it is generally safe to select "Use Git from Windows Command Prompt." This selection extends compatibility for Git commands to the Windows Command Prompt. We will not be using this feature for the sake of these instructions, but it is useful for users who prefer using the built-in Windows command line interface. Go ahead and **select either the first or second option, then click the "Next" button**.

5 Line Breaks: Three more options are presented on the second page. This time, the options affect how line breaks are treated in text files. Windows treats line breaks differently from other operating systems, so some editors such as Notepad will have trouble reading line breaks in files from other operating systems. Git provides a feature which can automatically convert other line breaks into Windows-style line breaks when files are retrieved from a repository. It will also convert them back before committing changes to the repository. There are very few instances where this feature is not desirable, so it is safe to **leave the first option selected and click the "Next" button**.



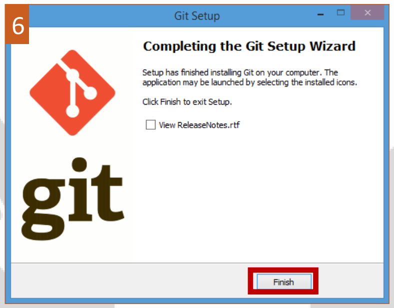
Configuring the line ending conversions
How should Git treat line endings in text files?

(●) **Checkout Windows-style, commit Unix-style line endings**
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

( ) **Checkout as-is, commit Unix-style line endings**
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

( ) **Checkout as-is, commit as-is**
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

http://msysgit.github.io/



Completing the Git Setup Wizard

Setup has finished installing Git on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

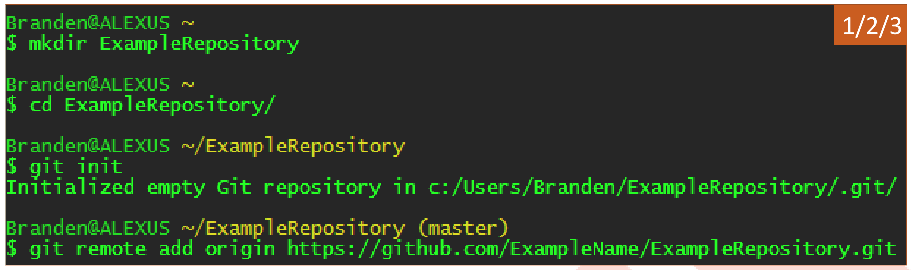
☐ View ReleaseNotes.rtf

6 Finish: Deselect "Open ReleaseNotes.rtf" on the final page and **click the "Finish" button**. Git and its primary components should now be fully installed and ready to run.

Initializing a Workspace

1 Create Folder: A folder must be setup locally to read, create, and modify files on the repository. This folder is commonly referred to as the **workspace**. From the command line, we can use the **`mkdir`** command to make the workspace folder. On Git Bash, make sure you are in the folder where you want to create the workspace. For most, the home folder is a good folder to choose. **Once you are in the correct folder, type `mkdir <name>`** where <name> is the name of the workspace folder (usually the repository name). Next, **open that folder with the command `cd <name>`**. Notice that the path above your command line changed to the path of the workspace.

2 Initialize Git: To designate the current working folder as a git workspace, simply **type the command `git init`**. If the Windows option to show hidden files and folders is turned on, you should notice that a new folder named ".git" was created in the workspace. This folder contains all of the Git information in the workspace. If you ever want to remove the workspace designation, simply delete the .git folder (try the command **`rm -r .git`**). If you cannot see the .git folder in your workspace, try using the command **`ls -A`**.

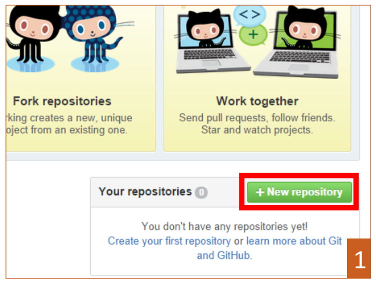


```
Branden@ALEXUS ~
$ mkdir ExampleRepository

Branden@ALEXUS ~
$ cd ExampleRepository/

Branden@ALEXUS ~/ExampleRepository
$ git init
Initialized empty Git repository in c:/Users/Branden/ExampleRepository/.git/

Branden@ALEXUS ~/ExampleRepository (master)
$ git remote add origin https://github.com/ExampleName/ExampleRepository.git
```

You may also notice that "(master)" was added after the path above the command line prompt. This tells us that we are on the branch named "master." We'll cover more on how to use branches later.

3 Add Remote: Lastly, we need to add the GitHub repository to the workspace as a **remote**. A remote is a reference to a repository which can be used in the workspace. The primary remote for a workspace is usually called "origin," so we can **use the command `git remote add origin <HTTP>` to add our new remote named "origin"**, where <HTTP> is the HTTP address of the repository we recently created. In this instance we are calling this remote "origin" because it is the standard, but any name can be used in the command when adding a new remote. SSH addresses can also be used in place of HTTP addresses, but they require some additional setup.

Creating a Repository


Fork repositories
king creates a new, unique
oject from an existing one.

Work together
Send pull requests, follow friends.
Star and watch projects.

Your repositories 0 | + New repository

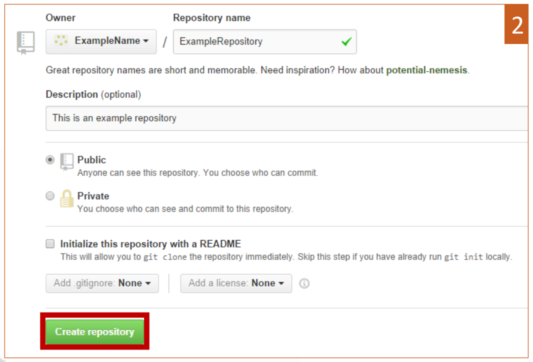
You don't have any repositories yet!
Create your first repository or learn more about Git and GitHub.

1

1 **New Repository:** We need to create a repository to work with on GitHub. If you are logged in on https://github.com/ you'll see a series of tutorials on the front page. These can be very useful for beginners, but for now just **click the "New repository" button**.

The following page requests a name and description for your repository. These fields are fairly straightforward and they can be almost anything you'd like as long as the name is not taken by another user.

2 **Configure:** Next it asks whether or not the repository is public or private. Private repositories will only be visible to invited users and require either a payed plan or for you to register your account as an educational one using a valid university email address. The remaining fields can be ignored. **After completing the form, click on the "Create repository" button**.


2

Owner: ExampleName / Repository name: ExampleRepository ✓

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

Description (optional): This is an example repository

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

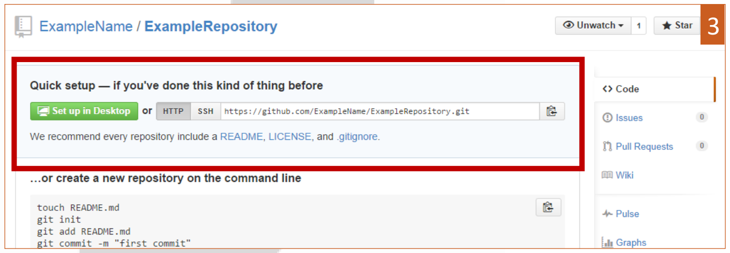
Private — You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: None | Add a license: None

Create repository

3 **Get Repository Link:** When you click the button, GitHub will bring you to your empty repository page, which provides many suggestions for getting started. The most important part of the page for now is the blue box which provides the HTTP and SSH addresses of the repository. These addresses will allow us to connect to the repository using Git locally. For these instructions we will be using HTTP addresses, which require a name and password any time a connection is made between Git and GitHub. For information on how to setup a repository with SSH links instead, visit https://help.github.com/articles/generating-ssh-keys/.


3

ExampleName / ExampleRepository — Unwatch 1 | Star

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTP | SSH | https://github.com/ExampleName/ExampleRepository.git

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
touch README.md
git init
git add README.md
git commit -m "first commit"
```

Using the Command Line

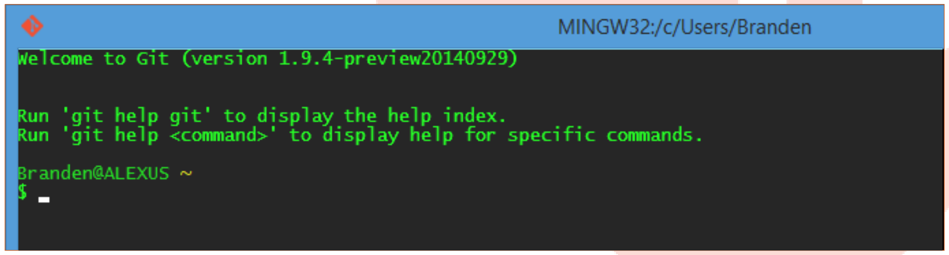
The **command line** is a simple interface for executing tasks on a computer. Instead of interacting with pictures and buttons, a user simply types a command and presses the "Enter" key. Of course, it is impractical to do everything in a command line as it requires memorization of commands and can sometimes be much slower than its counterpart, the Graphical User Interface (GUI).

Unfortunately, while GUI software for using Git is available, some options are daunting for beginning users while the others simply do not offer more than the most basic features. This booklet will focus on using the command line interface for Git since it allows for all of Git's features to be used without presenting too much information for basic users to be comfortable with.

If a GUI is still preferable, the most popular options are GitHub for Windows by GitHub (available at https://windows.github.com/) and SourceTree by Atlassian (available at http://sourcetreeapp.com/).

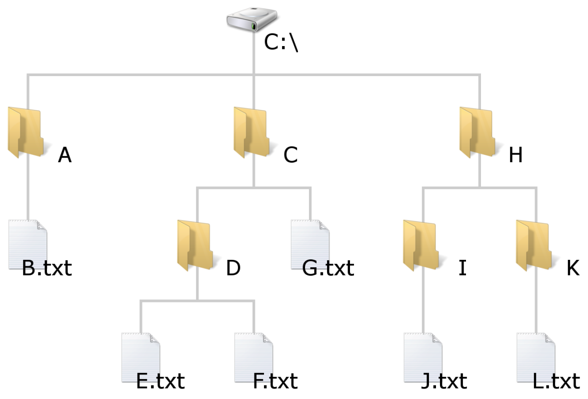
When Git was installed, a command line software called **Git Bash** was installed with it. Git Bash emulates the popular "Bash" command line interface used on many other operating systems. Although Git is usable in the Windows Command Prompt, Git Bash is typically the preferred interface. The Git-specific commands are almost identical between the two interfaces, but the more basic system commands can be very different. For more information on how to use the Windows Command Prompt, check out http://dosprompt.info/.

Try opening Git Bash to see the basic command line interface. After the introduction text, you should see a line which shows the Windows username of whatever account is logged in, an "@" symbol, and the name of the computer. In the example image, the username is Branden and the computer's name is Alexus. Following is a "~" or a *tilde* character. It represents the folder which is currently open. More specifically, it represents the "home" folder.


MINGW32:/c/Users/Branden

```
Welcome to Git (version 1.9.4-preview20140929)


Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Branden@ALEXUS ~
$ _
```

Folders (sometimes called directories) organize a computer's file by grouping them and nesting within other folders. Root folders in Windows are represented by drives. On most systems, the root folder or drive will simply be labeled "/c." The Windows Explorer labels this as "C:\"



Paths are addresses used to show where a file or folder is located. They list every folder between the root and the specified file or folder separated by a slash. For instance, the absolute path for J.txt in the diagram would be "/c/H/I/J.txt." If a path begins with a slash, it is called an **absolute path** and always starts at the root directory. Without a starting slash, it is a **relative path** and depends upon the current folder. From H, the relative path for J.txt would be "I/J.txt."

Quotes are not normally necessary to denote a path, but if any file or folder has spaces in it, a path must be wrapped with quotations. This can be avoided by using underscores (or "_") instead of spaces when naming files and folders.

One folder will also be set as the **home folder**, represented by the "~" character. For Windows, the home folder defaults to "/c/Users/<Username>" where <Username> is the account name of the user who is currently logged in.

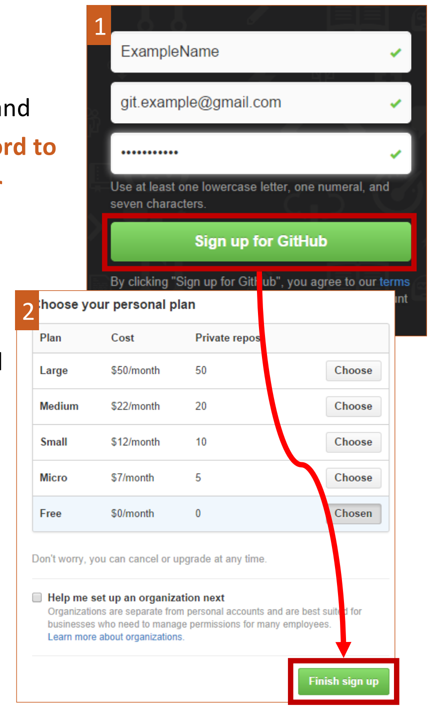
Interacting with files and folders in the command line is very similar to using the Windows Explorer program. To see the contents of the folder you're in, type the command **`ls`** (remember to press enter to submit a command). To change folders, type **`cd <path>`** where <path> is either the relative or absolute path of the folder you want to go to. Additionally, you can type **`cd ~`** to go to the home folder or **`cd ..`** to go up to the parent folder. Other system commands are given by the table below.

`pwd`	Display the path of the current working folder
`ls`	List all files and folders in the current working folder
`cd <path>`	Change the current working folder
`mv <path> <path>`	Move a file or folder to another path
`cp <path> <path>`	Copy a file or folder to another path
`rm <path>`	Remove a file or folder (must provide –r for folders)
`mkdir <path>`	Make a new folder
`exit`	Close the command line interface

Registering for GitHub

1 Sign Up: To use GitHub, you must have a registered account. **Navigate to GitHub at** https://github.com/ in your preferred browser and **submit a username, email address, and password to create new account, then click the "Sign up for GitHub" button.**

2 Select Plan: The next page will ask you to select an account payment plan. The free plan allows for unlimited public repositories and some private repositories for students and educators with valid email addresses. For most users, the free plan is enough. Make sure that "Help me set up an organization next" is not checked and **click the "Finish sign up" button**.



1

ExampleName ✓

git.example@gmail.com ✓

•••••••••••• ✓

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

Sign up for GitHub

By clicking "Sign up for Git ub", you agree to our terms

2

hoose your personal plan

Plan	Cost	Private repos	
Large	$50/month	50	Choose
Medium	$22/month	20	Choose
Small	$12/month	10	Choose
Micro	$7/month	5	Choose
Free	$0/month	0	Chosen

Don't worry, you can cancel or upgrade at any time.

☐ Help me set up an organization next

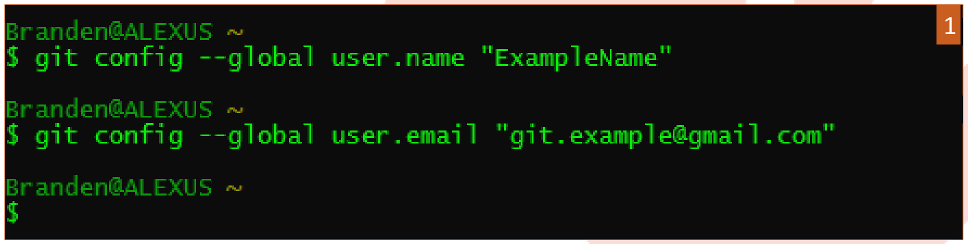
Organizations are separate from personal accounts and are best suit d for businesses who need to manage permissions for many employees.

Learn more about organizations.

Finish sign up

Configuring Git

1 Configure: Before using Git, you must configure Git to use your name and email address for GitHub. You can do this by opening Git Bash and entering the commands **`git config --global user.name "<Username>"`** and **`git config --global user.email "<Email>"`** where <Username> and <Email> are the username and email address used to register with GitHub. Be sure to note that there are two dashes before each "global."



1

```
Branden@ALEXUS ~
$ git config --global user.name "ExampleName"

Branden@ALEXUS ~
$ git config --global user.email "git.example@gmail.com"

Branden@ALEXUS ~
$
```