FCPA 2022  
  
Git and GitHub

Student Workbook 11

January 24, 2022

Author

Paul Kimball  
Interface Associates

1. Git and GitHub

Precious Source Code

## There are many types of source code

* + Program code (C, C++, FORTRAN, COBOL, Python, JavaScript, etc.)
  + Configuration files (JSON, XML, YAML, etc.)
  + Shell scripts
  + Documentation (Text, Markdown, HTML, PDF, images, etc.)
  + Default data files/databases
  + Files to control the build procedure (Makefiles, etc.)
* It takes real human effort to create source code
  + Time spent thinking
  + Time spent writing
  + Time spent testing
  + Time spent fixing and managing it
  + Time spent cussing and Googling
* Source Code is precious

Protecting your Source is Hard

* Backing up your files becomes important
* It's hard to keep track of all the changes you make as you write and debug code
* There are always more and more files!
  + Development tools add files that become part of your project
  + Compiling and linking creates even more files
* Traditional sharing methods are clumsy or lose vital information
  + Messaging systems: Zoom chat, email, webex ?
  + File-sharing systems: Dropbox, Google docs ?
  + You'll spend a lot of time zipping/unzipping
  + Which files are the most recent ones? They have the same names!

# Keeping Track of Source Code

* You *must* save it somewhere safe, but accessible
  + Network file access + off-site backups
  + **Source Code Management is important**
* If you change it, you *must* remember what you changed
  + You might have to put it back the way it was ("revert")
  + Saving multiple copies of *even one file* is tedious!
  + **Version Control is important**

## It becomes complex when you edit many files

### A single change in functionality can require edits to several source files (.h, .c, .cpp) that must be committed as a unit

#### It's easy to forget what you did

#### It's easy to accidentally delete something you cared about

#### It's easy to mix up your changes and finally produce a code base that does not build, or does not work properly

* Complexity grows as many people work on the same code
  + You *must* divide source code into many small files to encourage independent development
  + You *must* merge updates made by many programmers

## Source Code Management / Version Control Systems can help

### The two terms are often used interchangeably these days

* Many, many products

### IBM Rational ClearCase, Apache Subversion, Microsoft Team Foundation Server, Git, sccs, rcs, etc.

# Git

## Git is a Distributed Version Control System (VCS)

### Latest and most fashionable in a long history of tools

* Free, open source
* Fast and lightweight

## Very popular

* Written by Linus Torvalds, who wrote the Linux kernel

What Git Does

## Tracks the changes made to your source files

### **What** was changed (down to individual characters)

### **Who** did it

### **When** and (hopefully) why they did it

### **Which** files were changed together as a unit

* Saves project history in a "repository"
  + A database of tracking information
* Git can run on your workstation

### You don't need a network to manage your local repository

## Git can run "in the cloud"

### You can copy your code to a remote repository on a GitHub server

# Local Repository

## Local Repository

* + A collection of related files tracked by git

### On your local machine, in your file system

### Git stores the project history in a subdirectory named .git

# Remote Repository

## Same as a local repository, but "hosted in the cloud"

### You can back up your repositories so you don't lose them

### You can share your repositories with other developers

### You can browse open-source code any time you want to find examples and ideas

### You can integrate with bug reporting and issue tracking systems like Bugzilla, Jira, etc.

## Some Git popular Git hosting services

### GitHub

### Gitlab

### Bitbucket

### SourceForge

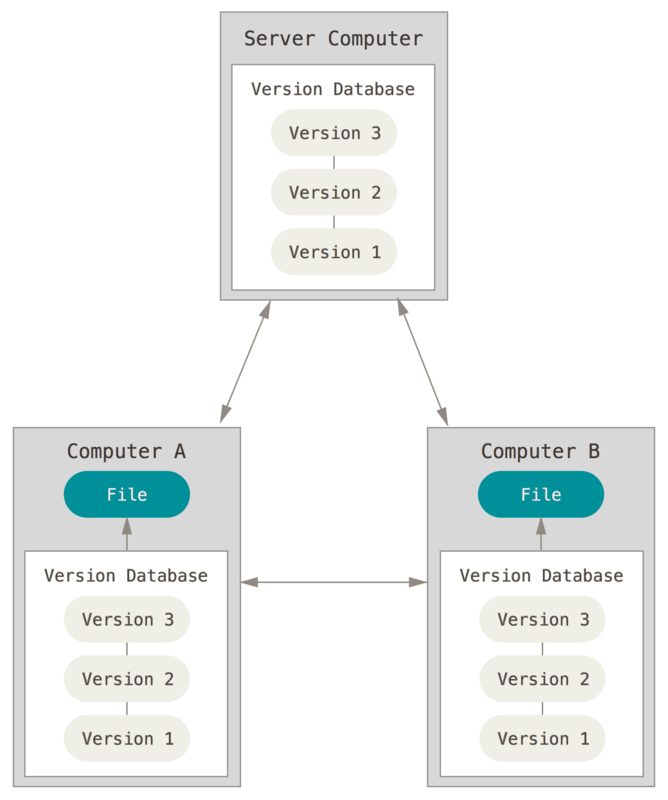
### Microsoft Azure DevOps

### Amazon AWS CodeCommit

### Corporate servers, e.g. github.ford.com

# Git is Distributed

* Git works on a local repository
* Git repositories can communicate with each other
  + When files are pushed, pulled, or cloned, they also bring their history
  + By default, a repository remembers *all* the historical changes to the files it holds



* All repositories are equal

### It's easy to clone an existing repository

### It's easy to publish ("push") your repository to the web

### It's easy to update your repository ("pull") to match another one

## No Central Point of Failure

### BUT. No Central Point of Control, either

## Big benefits

### Encourages sharing and modifying code

### Encourages collaboration

### Encourages multiple copies of repositories, so less chance of losing things

### Encourages you to dare mighty things, knowing you have a safety net

## It doesn't run itself

## You have to use it diligently

Git is your Safety Net

## You can restore the source code to any point in project history

### Revert changes if they don't work out

### Remove unneeded code without worry - you can always get it back

## You can do new feature development on an existing project without messing it up

### You get your own copy of the source code files and the history

### You can feel free to experiment with new code, even major refactorings

## You can compare code in your repository with code in other repositories

## You can merge changes from multiple developers back into a project's source code

### Git can tell if the changes conflict and lets a human intervene

Basic Workflow

* Developer "clones" or "pulls" the latest files from a remote repository to their local machine
* Developer makes changes and tests their work
* Developer makes a local "commit" of the changed files
  + The commit includes all files related to the change
* Developer repeats this cycle locally until they are ready to share their changes with other developers
* Developer "pushes" changes back to a remote repository or provides a patch file to another developer

# Git is Very Flexible

* There is no industry-standard "project" structure

### IDEs like Eclipse and Visual Studio provide a template, but this is only a starting point

## A Git project is just a directory + its contents

* + including subdirectories, recursively
* YOU decide which files are precious "Source Code"
  + You can tell git to ignore files you don't care about

Compiled artifacts and generated code can be re-created from source code; however, they might be archived for distribution or legal purposes

VS .sln and .vcxproj files might be important to you, or might be just a silly detail of the editor

* YOU tell git when to save your changes
  + Modified files are not committed automatically

# Downloading Git

* <https://git-scm.com>
* Binary distributions

### <https://git-scm.com/downloads>

* Source code:

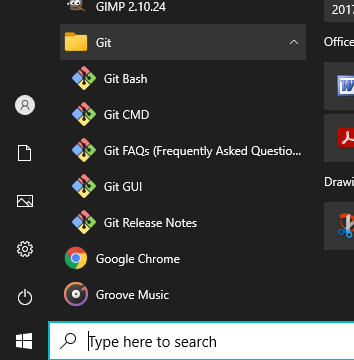
### <https://github.com/git/git>

* Git for Windows
  + https://gitforwindows.org/
  + A port of git to Windows, which includes several UNIX tools, including the "bash" command shell

# The git Command

## We'll use git from the command line shell

* + Simple, portable
  + You can start a shell from the Windows Start menu - The Git folder has shortcuts to make sure that git is in your %PATH%
* Git-Bash will be used in the next few sessions
  + After that you can use the Windows Command Prompt if you want
* The git command is usually followed by an *action*, and possibly some *flags*



**git** *action --flag flag-value*

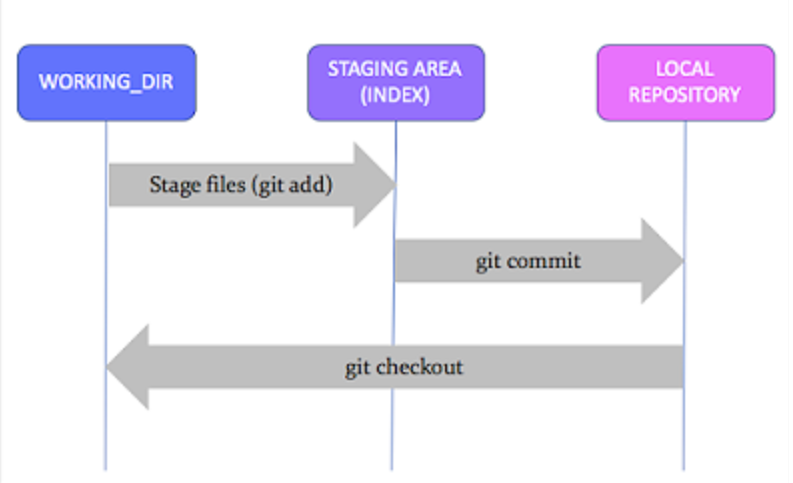
* Run the following command to check the version of Git installed

$ **git --version**

git version 2.8.1

Git Conceptual File Areas

* Working Area



* + The folder(s) where you are currently working on source code
  + Files here are "untracked" files
* Staging Area or "Index"
  + Temporary area for files that you'll save to the Repository
  + **Important - Lets you save multiple files as a single transaction**
  + Files here are "staged"
* Local Repository
  + Holds the record of your committed changes
  + It is the area where everything is saved
  + Files here are "committed"

Git Configuration

* The global configuration holds settings for all of your repositories
  + Stored in <home\_dir>/.gitconfig
* Local configuration controls a single repository
  + Stored in <project-dir>/.git/config
* Set the user name

$ git config --global user.name "Paul Kimball"

* Set the user email

$ git config --global user.email "pkimball@example.com."

* List configuration values

$ git config --list

# Your GitHub account

## Allows you to create and manage remote git repositories

## Lets you push files from local -> remote

## Lets you pull files from remote -> local

## Lets you compare your local repository with a remote copy

# GitHub Authentication

## On your own machine, in your own directories, you can do whatever you want with Git

### Create repositories

### Add and remove files

### Commit and roll back changes

## When you use GitHub, you are subject to:

### Authentication

#### Prove who you are

### Authorization

#### Knowing who you are, here's what you can do

## To access GitHub via a web browser:

### Use your User ID / Password to authenticate

## To access GitHub from the command line:

### You will need a Personal Access Token (PAT)

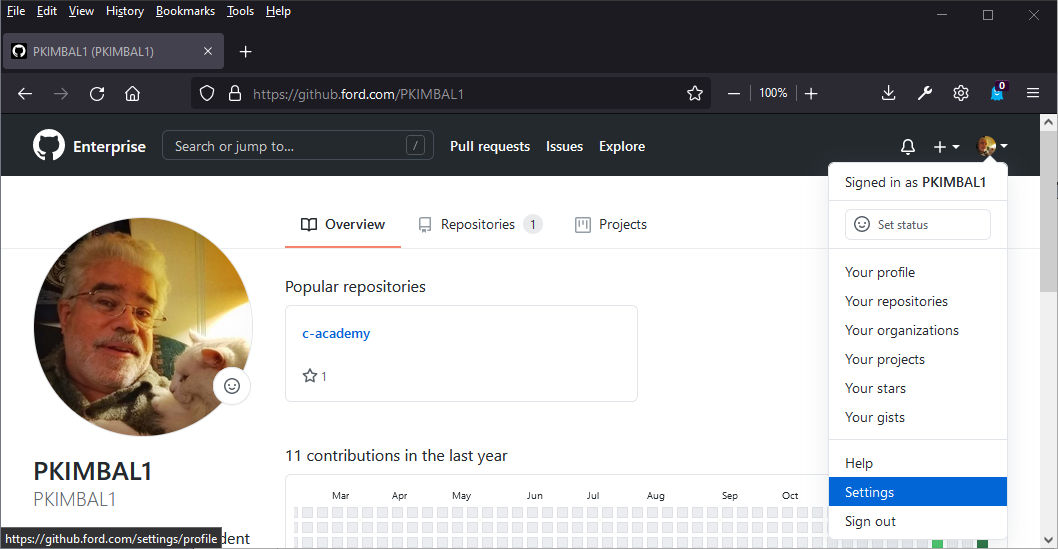
#### The PAT authorizes the git command to work as your agent

### You must log in to GitHub to generate a PAT

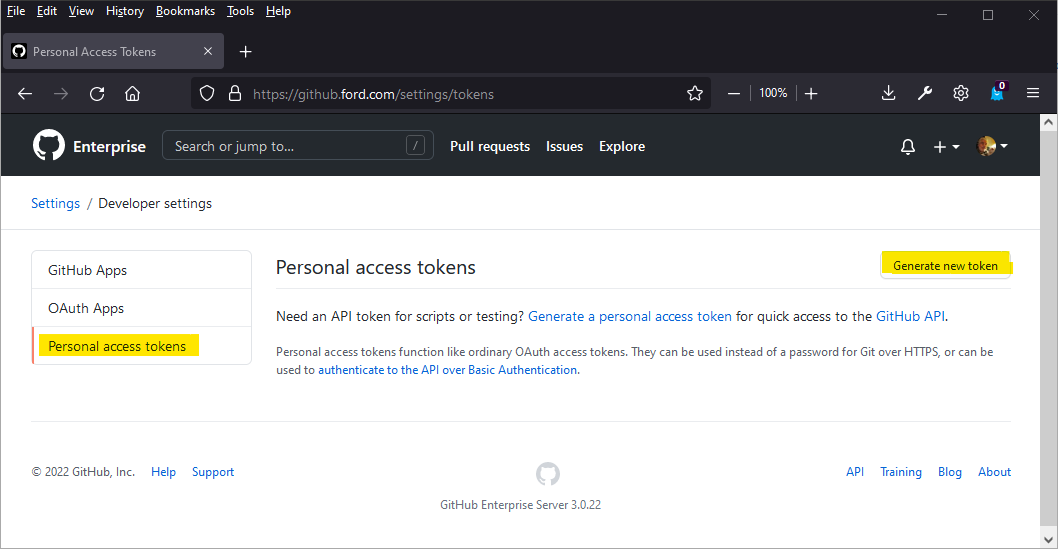
#### The token is good until it expires or is revoked

# Generate a PAT

## Log into GitHub and go to "Settings":



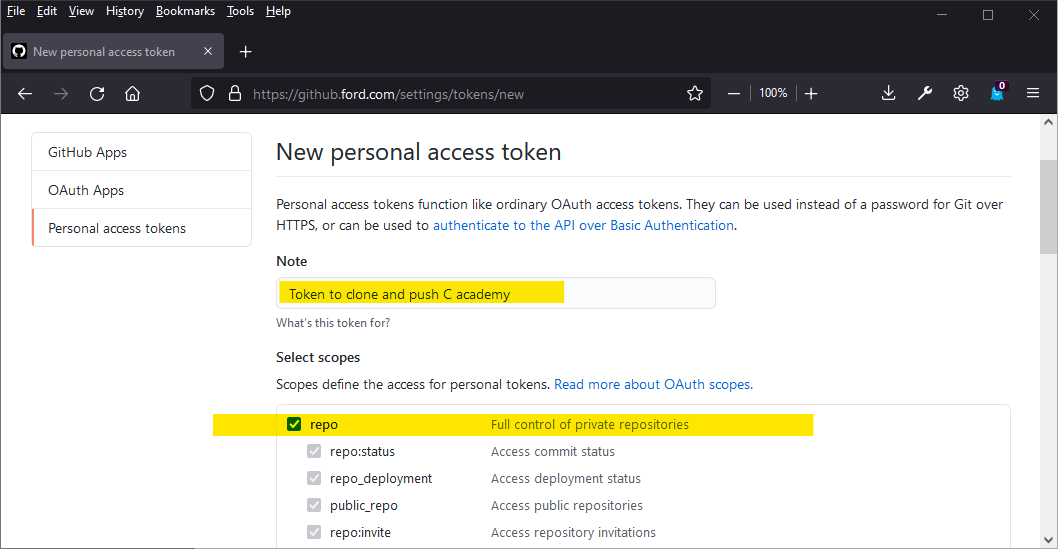
## Then select "Developer settings"



# Generate a PAT

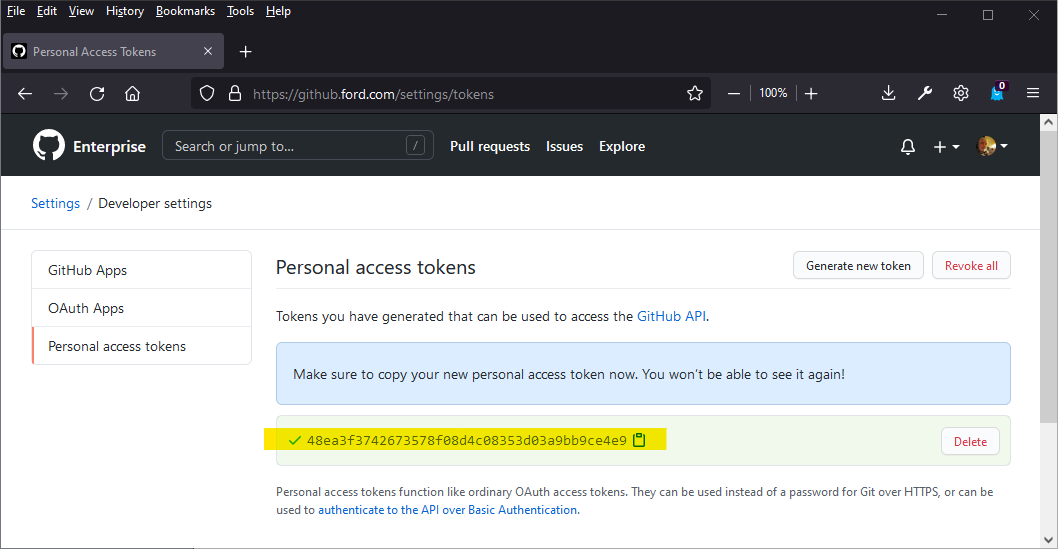
## Enter a note and select "repo" scope for authorization

### Then scroll down and click button to create token



## Make sure you copy it while you can and put it somewhere safe

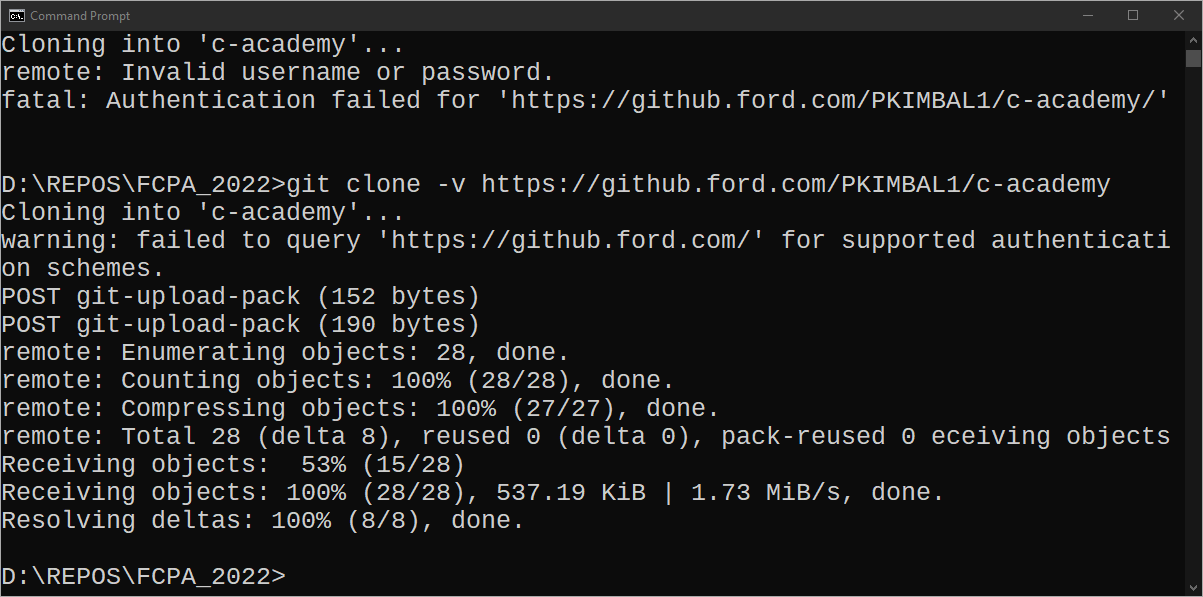
### You can click the little clipboard icon to copy



# Using your PAT

## Later on, when you use the git command to clone a repo:

## 



# Cloning a Repository from GitHub

## See cheat sheet

# Create a Repository

## See cheat sheet

# Add New Files

## See cheat sheet

# Change Files

## See cheat sheet

# Stage Files

## See cheat sheet

# Commit Your Changes

## See cheat sheet

# Git config files

## <https://itknowledgeexchange.techtarget.com/coffee-talk/files/2020/08/git-config-file-locations.png>

## git config --list --show-origin

# GitHub and Local Repositories

## GitHub First

### Log in to github

### Create empty repo on github

### Use github project URL to clone repo to local machine

### Git remembers the upstream repo is on github

### Add files and commit to local repo

### Push to remote repo whenever you want

### Use git -diff to see if your local repo is out of sync

## Local Repo First

* + Create and init repo on local machine
  + Add and commit files to local repo
  + Set upstream repo in .git configuration

### Push to remote repo whenever you want

### Use git -diff to see if your local repo is out of sync

# Git is Not Magic

## Git does one job - it tracks changes to your files

### Git doesn't know whether you're writing programs or fanfiction

### Git doesn't know about your project "structure" or "type"

## You have to tell git what to do

### Git doesn't create source files or VS projects for you

### Git doesn't "automatically" save your changes

## You can add git to an existing project

### But this won't "automagically" know file history from before

## You can remove git from a project

### But now it's gone, and its history with it

## You can delete your git repository and all your files

### And they're gone. Sorry.

#### You must have backups!!

## You have to tell git which files to manage

### Git will happily manage files that you don't want it to

#### You must have a .gitignore file (!)

### Git will happily ignore files you thought were important

#### You must remember to add AND commit files

## Git works best with human-readable text files

### There are better tools to manage music, video, photos, etc.

## You must use git continuously, conscientiously

### Don't just leave and go home, and hope that git remembers where you were and what you were doing

### If you lie to git, it will believe you