TAM Bootcamp - Git

Login to gitlab.eng.vmware.com

- Use your VMware AD credentials

-or-

Login to github.com

- Use your personal account credentials

Getting Started (Install Git)

On Windows (if you are a windows user)

- Open a terminal session
- Install git via the procedure here
  - http://git-scm.com/download/win

On a Mac (if you are a mac user)

- On your laptop (this example is on the mac)
  - Open a terminal session

- Install homebrew (if you don't have it installed) on your mac
  - You can get the instructions here:
    - https://brew.sh/
  - These are also the commands for brew:

    % brew update

    % brew doctor

- Install/upgrade git on your mac
  - Enter this command

    % git –version
  - You should see something similar to:

    Git version 2.24.1 (Apple Git-126)
  - If you see this, you are running the apple version of git, not the official distribution
  - You may even get the following nasty-gram from github

    Hi @davemazur,
    You recently used a password to access the repository at davemazur/velero_dev_guide with git using git/2.24.1 (Apple Git-126).
    Basic authentication using a password to Git is deprecated and will soon no longer work. Visit https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information around suggested workarounds and removal dates.
    Thanks,
    The GitHub Team
  - Next, install the git client with this command:

    % brew install git

- Brew installs the official git distro in /usr/local/bin
- So far, so good. Now you have to change your path to use the official git distribution
- Now enter this command:
  % export PATH=/usr/local/bin:$PATH
- Confirm that the git version is correct by entering this command:
  % git –version
- You should see that the version of git has been upgraded
  Git version 2.30.0

- In order to upgrade git (if you need to in the future)
  % brew upgrade git

In a browser, go to gitlab.eng.vmware.com (or your github.com account). We are using github.com for this example:

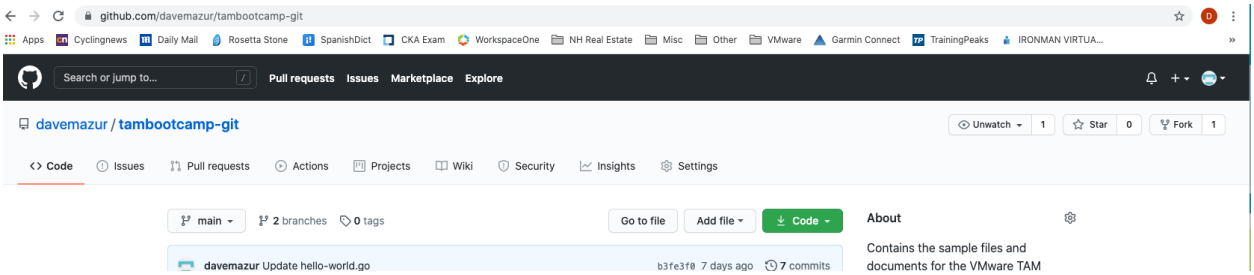
- Go to https://github.com/davemazur/tambootcamp-git
- Both github and gitlab have the same content

Basic Commands

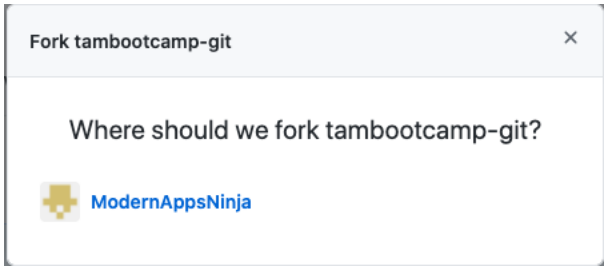
Fork the tambootcamp-git repository

(Fork creates your own copy of the repository that you can use for the rest of the lab)

- Login to your github.com repository
- Go to davemazur/tambootcamp-git



- In the upper right hand corner, select “Fork”
- When it asks you where to fork tambootcamp-git, select your personal repository



- Now you should see “tambootcamp-git” under your repositories

Clone the sample project in your repository onto your laptop

- On your laptop, create a directory and cd into the directory

```
% cd /users/<your home directory>
% mkdir github.com
% cd github.com
% mkdir <the name of your github account>
% cd <the name of your github account>
```

- Now you will make a local copy of the tambootcamp-git repository
    - Enter the git clone command (your command will vary)

      % git clone https://github.com/<the name of your github account>/tambootcamp-git.git

    - You should see output similar to:

      ```
      Cloning into 'tambootcamp-git'...
      remote: Enumerating objects: 6, done.
      remote: Counting objects: 100% (6/6), done.
      remote: Compressing objects: 100% (5/5), done.
      remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
      Receiving objects: 100% (6/6), done.
      ```

    - Change directory into the newly created tambootcamp-git

      % cd tambootcamp-git

    - Look at the contents of the directory

      % ls -al

    - Do a git status. Git status will be used often to provide information on your current state

      % git status

      ```
      On branch main
      Your branch is up to date with 'origin/main'.

      nothing to commit, working tree clean
      ```

Modify/Checkin a file

(The following few commands (status/add/commit/push) show you how to make changes to a file and check them into your repository in git)

- Let's edit a file using the "vi" editor (if installed). You can use any editor to make changes to the file

  % vi hello-world.go

- Next, make a change (any change you want) in the file

```
# file:  hello-world.go
# description:  my very first go program

package main
import "fmt"
func main() {
   fmt.Println("hello world")
}
```

- Save the file with :wq

Git status

- "git status" provides you with the state of the current git commands
- Enter the following command:

```
% git status
```

- Git then gives you the following output:

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
modified:   hello-world.go

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

- The output tells you that the file, hello-world.go, has been modified. It also tells you that you can use "git add hello-world.go" to update the file

Git add

- The "git add" command prepares your file for checkin to the repository
- Enter the following git commands:

```
% git add hello-world.go
% git status
```

- You should see output similar to:

```
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
modified:   hello-world.go
```

- This tells you that your changes need to be "committed"

Git commit

- Now that you have added your file, you need to “commit” the file
- Enter the following command

  % git commit -m "added notes to hello-world.go"

- You will see output similar to:

  ```
  [main b7150cf] added notes to hello-world.go
   1 file changed, 3 insertions(+)
  ```

- Next, let’s get the status again. Enter the following command:

  % git status

- You will see output similar to:

  ```
  On branch main
  Your branch is ahead of 'origin/main' by 1 commit.
    (use "git push" to publish your local commits)

  nothing to commit, working tree clean
  ```

- This is telling you that you need to use “git push” to publish your changes to hello-world.go

Git push

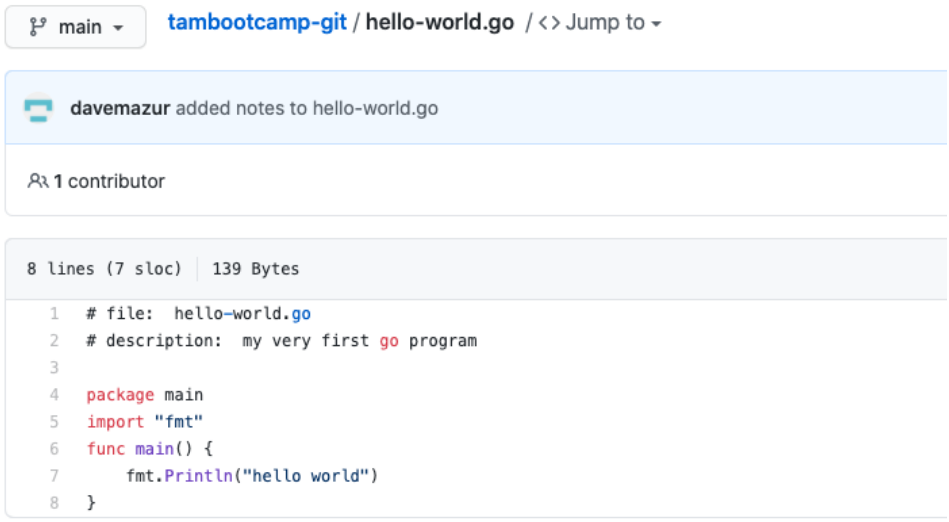
- Git push takes your changes to hello-world.go and pushes them up to the repository
- Enter the following command:

  % git push

- You will see output similar to the following, which tells you that the changes to the file have been published to your repository:

  ```
  Enumerating objects: 5, done.
  Counting objects: 100% (5/5), done.
  Delta compression using up to 8 threads
  Compressing objects: 100% (3/3), done.
  Writing objects: 100% (3/3), 411 bytes | 411.00 KiB/s, done.
  Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
  To https://github.com/davemazur/tambootcamp-git.git
    818263e..b7150cf  main -> main
  ```

- Go over to the github.com webpage and look at the changes



- You should see the line(s) that you had changed on your laptop are now pushed to the repository

Git restore

- Git restore can be used to recover a file if you accidentally delete a file in your local directory
- First, delete the hello-world.go file from your laptop

  % rm hello-world.go

- Now, let's use git to recover the file using the following command:

  % git restore hello-world.go

- List the files in the directory to show that hello-world.go has been restored

  % ls

- If you want to restore a previous version of a file from the master branch, you can use the following command (note the ~2 to specify previous versions)

  % git restore --source main~2 hello-world.go

Git log

- The git log command gives you detailed log information on the commands that you have used
- Enter the following command:

  % git log

- You should see output similar to:

  commit b7150cfac7fb1f216f5eaffc92ea6040f69bea6a (**HEAD -> main**, **origin/main**, **origin/HEAD**)

Author: Dave Mazur <davemazur@comcast.net>
Date: Tue Jan 26 13:37:57 2021 -0500

added notes to hello-world.go

commit 818263e5f2395b9adbc29f98ad4f83c754b7baff
Author: davemazur <davemazur@comcast.net>
Date: Tue Jan 26 13:29:06 2021 -0500

Create hello-world.go

commit fb2bab0aa08152172aa6aecb37afa1028405b7e0
Author: davemazur <davemazur@comcast.net>
Date: Tue Jan 26 13:25:27 2021 -0500

Initial commit

Git help

- The git help command give you help on the various git commands
- Try entering the following commands

% git help git

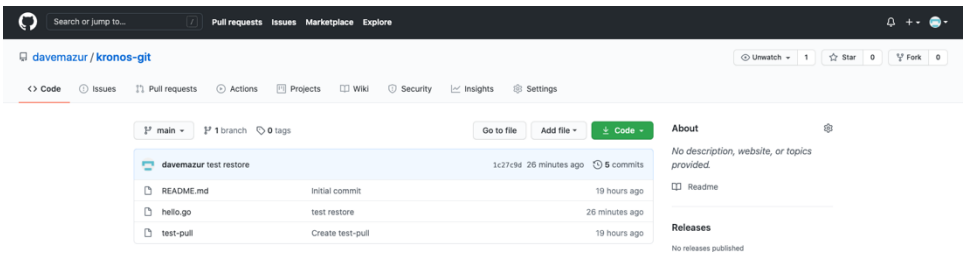
% git help -a

% git commit --help | more

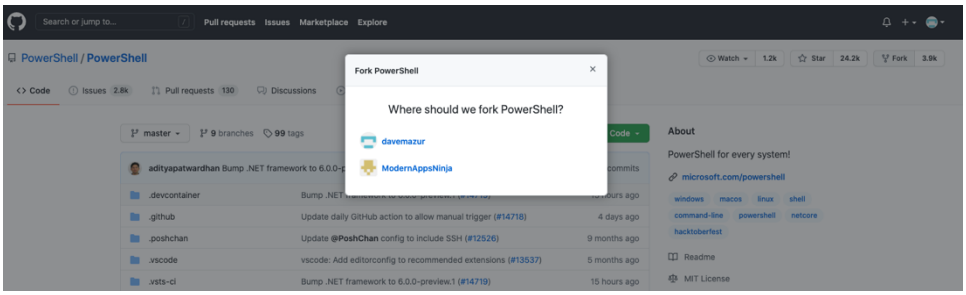
Advanced Topics

Git fork

- The git fork command makes a copy of another users repository and places it in your repository. The fork is a completely separate copy (i.e. you do not get any changes made in the original repository)
- Go to the github.com website



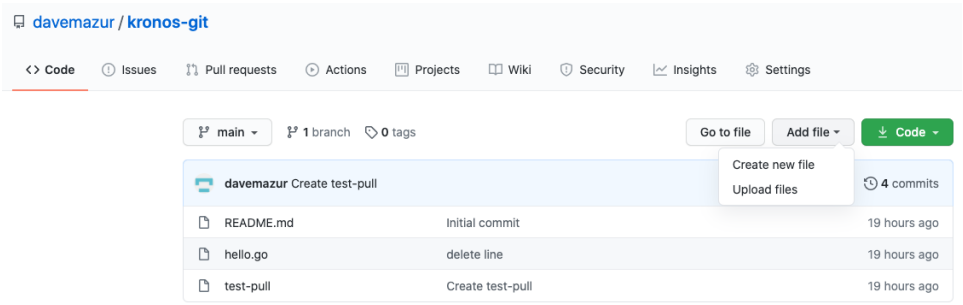
- Search for the "powershell" repository
- In the upper right hand corner select "fork"
- Select where you want to fork the repository



- Look at your repositories to view the new fork
- You can now do a “git clone” from your forked repository onto your laptop

Git pull

- When two or more developers are working in the same code base, they are adding/deleting/modifying the same or different files. You will want to pull down these changes during the day to make sure everyone on the team is using the same codebase for development.
- In this example, Developer 1 adds a new file to the tambootcamp-git repository. Let’s simulate adding a new file into the repository:
    - Use the "Add File"->Create new file in the github web screen
    - Name the file “test-pull”
        - Add a comment to the file
        - Save the file (at the bottom of the page)
    - Look at repository in github and you should see that the file “test-pull” has been created:



- Now, you as Developer 2, wants to “pull” the latest changes from the repository into your local directory
- Do the “git pull” command below:

  % git pull

- If you do an “ls -al”, you should have pulled down the file that dev 1 created.
- Best practices for using git pull
    - Pull every morning
    - On very heavily used repositories, pull multiple times during the day

Branching

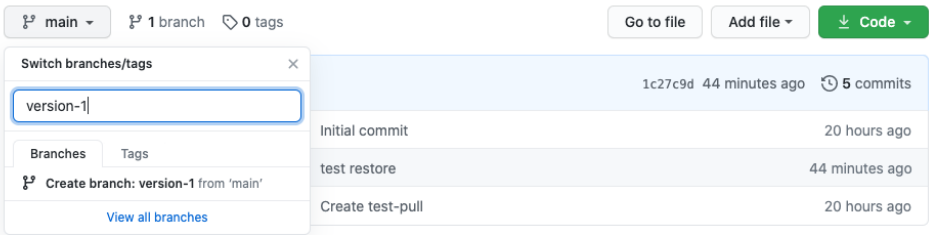
- When you create a branch, you create an identical copy of the project at that point in time

- Important Concept: When you do a “git pull” or “git clone”, behind the scenes, git pulls down ALL branches from the remote repository. These branches are now on your laptop and you will see how to switch between branches in the lab below.
- First, let’s explore the concept of remote branching and then local branching

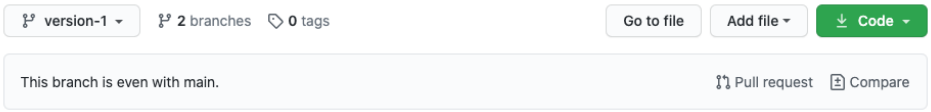
Remote Branching

- Remote branching occurs in the github.com repository (hence, being “remote” from your local laptop)
- You would use this for a project team...all team members work in this branch

  - Navigate to your repository in the github.com web page
  - Select the dropdown under the main branch



  - Create a branch called "version-1".
  - Now the number of branches changed to (2) and the default branch is "version-1"
    - The branches you have are “main” and “version-1”
    - You are automatically switched to the new branch, version-1
    - Any changes you make to the files in the repository will be applied to this branch



  - Go to your local repository on your laptop and get any changes using git fetch:
  -

    % git fetch

  - This updates the remote branch
  - Take a look at your available branches. The current branch is noted by an “*”

    % git branch -a

  - The output will be similar to:

```
  main
* version-1
  remotes/origin/HEAD -> origin/main
```

remotes/origin/main
remotes/origin/version-1

Local Branching

- This is a branch that only "you" can see. It only exists on your local machine
- Let's create a local branch:

  % git branch test-local-branch

- Let's view the current branches:
- % git branch

  - Note *main is the current context

- You are still on the main branch. Let's switch to the new local branch:

  % git switch test-local-branch

  - Note *test-local-branch is the current context
  - *Note: Git checkout -b <name> will create the branch and switch contexts*

- You can compare branches using the following command:

  % git diff <local-branch> <origin/remote-branch>

Merge changes with another developer

- There are times when two or more developers are working on the same lines of code in the same source file. You do not want to automatically overwrite changes that someone else has made. Git will first try to automatically resolve changes. If it cannot, you must manually merge those changes. Here's an example:
  - Developer 1 makes changes to hello-world.go in the remote repository in github web
  - Developer 2 (you) makes changes to hello-world.go in the local branch test-local-branch
  - Try to checkin the changes using the following commands:

    % git commit -a -m "checkin lines 3/4"
    % git push

  - You should see output similar to:

    To https://github.com/davemazur/tambootcamp-git.git
     ! [rejected] test-local-branch -> test-local-branch (non-fast-forward)
    error: failed to push some refs to 'https://github.com/davemazur/tambootcamp-git.git'
    hint: Updates were rejected because the tip of your current branch is behind
    hint: its remote counterpart. Integrate the remote changes (e.g.
    hint: 'git pull ...') before pushing again.
    hint: See the 'Note about fast-forwards' in 'git push --help' for details

- Note that the output is giving you a "hint" to do a git pull…Let's try that:

  % git pull

- The next part is not as obvious. Edit the file using the editor of your choice

  % vi hello-world.go

```
<<<<<<< HEAD
#line 3
#line 4
        # description:  my very first go program
=======
# add line 1
# add line 2
# description:  my very first go program
>>>>>>> e7141045aae5d31c0b3db9ecff8eb777b1d09e50

        package main
        import "fmt"
        func main() {
          fmt.Println("hello world")
}
```

- This shows you where the merge conflict occurred. I highlighted the two areas in red
- Using the editor, remove the lines you do not want and save the file
- Then follow the standard add/commit/push procedure to checkin the file. Make sure to add a descriptive comment to tell others which lines you changed and why

  % git add hello-world.go
  % git commit -m "fix merge"
  % git push

- Look at file in git repository
  - You should see the latest changes

Git Ignore

- As we have already seen, git constantly looks at the files in your directory to determine what needs to be pushed up to the remote repository. If it sees a new file, it tells you it is "untracked". The problem is that not all files should be pushed (i.e. log files, executables, credentials files". The .gitignore file allows you to specify which files to ignore

- Let's go back to your local repository

  % cd tambootcamp-git

- Create a file you do not want to checkin and put in some random text

% vi my-secret-file

- If you now do a “git status”, git will show the file as "U" - untracked

  Untracked files:
    (use "git add <file>..." to include in what will be committed)
  my-secret-file

- In that same directory create a .gitignore file

  % vi .gitignore
  **/my-secret-file/*

- Save the file
- Now do a “git status”. The output should no longer show “my-secret-file” as untracked.

End