Git Basics

Git Bash Command Console

No nano on Windows so have to use notepad instead

Create Repository

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git Bash Command Console

$ git --version

git version 1.9.5msysgit.0

**CREATE REPOSITORY:**

$ git init my\_first\_repository -- passed directory name

Creates new directory called my\_first\_repository

**CREATE a git repository inside the current folder:** CREATES .git file

$ git init

Initialized empty Git repository in~/project\_folder/.git/

$ ls –a to show hidden files

. .. .git this .git folder is the local repository

Name of repository is not essential

**.git folder keeps track of everything that happens**

**Can delete everything else in your project and if .git folder still existed have all the info needed to restore your work**

Rename the directory:

$ mv treehouse\_test\_project/ treehouse\_new\_test\_project

**Delete Repository:**

$ rm –rf name

Add File to Repository -- Committing changes

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Create README.txt file (with notepad) in the Project Directory (project\_one)

ADD NEW FILE to the repository: git add command [Add top repository so git knows to track it]

$ git add

$ git add README.txt

**COMMIT CHANGES**: git commit command -- opens a commit message template

$ git commit

Windows git editor: VIM:

Hit the Esc key; that goes into command mode. Then you can type

* :q to quit (short for :quit)
* :q! to quit without saving (short for :quit!)
* :wq to write and quit (think write and quit)
* :x to write and quit (shorter than :wq)
* :qa to quit all (short for :quitall)

INSERT MODE: a

COMMAND MODE: Esc

**ESTABLISH IDENTITY** on GIT with “git config” command to set email and identity options (using global flag)

Global flag – lets git know these changes apply to all our repositories

$ git config –global user.name “Daniel Niclas”

$ git config –global user.email “[dniclas@sbcglobal.net](mailto:dniclas@sbcglobal.net)”

Add more information:

$ vim README.txt

**WITH COMMIT MESSAGE THEN COMMIT**

$ git commit –a -m “Added project mission statement to README file”

**-a (all)** tells git to commit all the changes it can find

**-m (message)** allows us to provide the commit message to “git commit” command directly

**Staging Area**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

$ git status Shows the current version control status of our project

Only show information about files that have some special status to get

If file has already been added to the repository and it hasn’t been changes since we last committed it then there is nothing special about it

Add files: file1 file2 file2

Need to add to repository to be able to **TRACK CHANGES:** $ git add

$ git add file 1 moved it to the staging area

$ git add file2

$ git add file3

$ git commit –m “message”

Make changes to a file and check status $ git status the files are now being TRACKED and can detect changes

**Commit ALL:**

$ git commit –a (-a all flag) to commit everything that has changed

$ git commit –a –m “message” <<<

Separate commits:

$ git add file1 Added file1 to the staging area

$ git commit –m “message” commit it by itself

Git Commands

* git status - show the current status of the git repository, including if there are any uncommitted changes and whether or not any of our changes have been put in the staging area.
* git add - not a new command for this video, but we learned that it does something more than just add files to be tracked - it also adds changes to the staging area.
* git commit - not a new command for this video, but we learned that without any arguments (or the -a flag) it will default to committing everything that's currently in the staging area.

Git Commands

* git log - Show us a chronological log of all of our commits to the current repository.
* git checkout - "check out" a different version of the code from the one you're currently looking at.
* git diff - create a "diff" view to demonstrate what has changed between two different versions of your repository.

$ git log log of all the commits

Checkout:

$ git checkout 40894 use first 5 or 6 characters of the commit number

To get back to the top:

$ git checkout master

To investigate what is different or what has changed from one file to another:

$ git diff identifier1 identifier2

$git diff 03ce3c 408948

**Branching**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

MASTER BRANCH is the TRUNK

MASTER BRANCH is usually the one deployed in production -- the canonical version of the project

Create a new branch to work in:

$ git branch foo\_feature

**SWITCH BRANCH:** with branch name instead of commit identifier

$ git checkout foo\_feature checkout the latest commit of this branch (the head commit)

Make changes to a file and then commit the file

$ git commit –a –m “getting started on foo\_feature”

SWITCH back to master:

$ git checkout master

CREATED BRANCH and SWITCH to BRANCH -- $ git branch to create branch and $ git checkout to switch to it

COMBINE THESE

**CREATE BRANCH AND SWITCH TO IT** all at once using the –b flag -- create branch if it does not exist before checking it out

$ git checkout –b bar\_feature

Git Commands

* git branch - list all branches in the current repository and indicate which branch you're currently in.
* git branch -D branchname - delete the branch named branchname from the repository.

$ git branch **SHOW LISTING OF ALL BRANCHES**

$ git branch –D branchname DELETE BRANCH

**Merging**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Git Commands

* git merge branchname - merge the history from *branchname* into the current branch.

Merge changes from one branch to the one you are currently in (master):

$ git merge branchname

Merge Errors: merge issues due to differences in a file like README.txt

If different lines are edited GIT can accept the changes and perform merge

If auto-merging fails:

Simplest solution is to edit the conflicted file

Remove the conflict markers from the files in issue

Let git know issue is resolved:

**Add change to staging area and run git commit: For Merging Following Conflict Must do it SEPERATE**

$ git add file1

$ git commit

**Working With Remote Repositories [Push and Pull from Remote Repository]**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Cloning: Usually the first step to working on a git based project

Cloning creates a copy of the repo without changing it in any way

Cloning the repository to your local computer

And telling it about the remote repositories you need to work with

$ git clone network address if on another computer

$ git clone repository\_to\_clone name\_of\_new\_repository (2 args: repo to clone and new repo name)

Git has now automatically set up our original repo as a REMOTE repo when it created the new one

Git automatically names it “origin”

**ALL REMOTES AVAILABLE TO THIS REPO:**

$ git remote

Only the clone has a remote repo added

To **ADD A REMOTE** repo to the origin:

$ git remote add our\_clone project\_one name for: our\_clone

Path: local dir name

**Pushing and Pulling**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Send changes to the remote:

Git push: send all changes from current repo to the remote repo

If new branch created MUST call out the branch and provide branch name

$ git push origin branch\_name two args: remote name: origin “new\_feature” is the name of the new branch

Merges changes in from another repo

To pull from remote repo:

$ git pull origin new\_feature args: remote to pull from and what branch

**Git Flow**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

$ sudo apt-get install git-flow

Prep repository:

$ git branch integration

Initialize repository to get ready for git flow:

$ git flow init

Set up branch for integration of the next release: [integration]

**START WORK** on new feature:

$ git flow feature start feature\_name

**FINISH WORK** on new feature

$ git flow feature finish feature\_name

Merges feature branch into integration. Deletes the old feature branch and switches over to integration

Hotfix branches:

$ git flow hotfix start name

$ git flow hotfix finish name