**Session ID:** DEVWKS-1255. **Session Title:**Intro to Git

**A Brief Introduction to Git** <https://developer.cisco.com/learning/modules/programming-fundamentals/dne-git-basic-workflows/hands-on-with-git-and-github/>

Slide 1 Welcome everyone. When I say “Cisco Live” you say “Let’s Go”. Cisco Live/Let’s Go x 2

Slide 2 This is Intro to Git and I’m Alexander Stevenson, software engineer at Cisco

Slide 3 Cisco Webex App: chat with me after the session by following the instructions on this slide.

Slide 4 This session is built around this learning lab: A brief Introduction to Git, found at this URL

Slide 5 Agenda: Intro, What is Git?, Get familiar with the Learning Lab environ, Hands-on lab, Call to action.

Slide 6 Introduction: Git has been intimidating to me in the past. Not so much when I was first learning it, but when I had to use it at work for things of consequence. The good news is it has saved me many times.

Slides 7-8 The Git Guru: At the end of this session, you will know how to use ‘git checkout’ to change branches, create a branch and update a single file back to its last known good state

Slide 9 What is Git? Git was originally created by Linus Torvalds, who also made something you may of heard of called ‘Linux’. Anyway, Linus and saw the need for a comprehensive, and free, source control management system and the rest is history. "git" can mean anything, depending on your mood. “got it, thanks” “Global information tracker” or “growing infinitely tricky” all come to mind.

Slides 10

* Git manages files on your computer so that when you make changes, you can compare old and new versions of files. This aspect of file management is called Version Control.
* GitHub also enables you to coordinate work with others on new versions of files.

Slide 11 Objectives

* Understand what problems Version Control Systems help us solve.
* Learn and practice your first Git workflows.
  + - **Clone** our sample code in the browser development environment.
    - Create a "safe place" for you to make and **commit** your changes.
    - Learn how **to incrementally save** your code changes.

Slide 12 Getting familiar with the Cisco DevNet Learning Lab environment (go to URL)

Slide 13 Main parts of the LL environ: Table of Contents, Instructions, IDE (Integrative Development Environment) **Show features**

Slide 14 (Click ‘Start Learning’) Lab walk-through, File System and Editor, and Interactive Terminal. (**Show features**)

Slide 15 Hands-on Lab

Slide 16 Once again, this session is based on the DevNet Learning Lab ‘A Brief Introduction to Git’ and here is the URL again in case any of you cleared your browser cache in the last 5 minutes lol

Configuring Git Locally

When you make a commit (think save) to a Git repository, Git automatically includes) the name and e-mail address of the person that made the commit, along with a commit "message." Commit messages can serve as descriptive change logs that record the history of the changes in the project.

**Telling Git to default to "main" branch**

GitHub defaults to using the **main** branch on all new repositories created, so you want to set up your local Git so that when you type ***git init*** the branch created is also named main.

**git config --global init.defaultBranch main**

**Telling Git who you are (one-time setup)**

You need to tell Git who you are before you can commit any changes to a repository.

**git config --global user.name "Cisco Cisconian"**

**git config --global user.email cisconian@cisco.com**

Run this command to see the current configuration

**git config –list**

You should see your name and email, as well as init.defaultbranch=main

**Create a personal access token**

You can create a personal access token with repo permissions in the Settings of your personal GitHub account, under Personal access tokens. Then you would enter the token whenever you're prompted for a password.

Basic Git Workflows

**Cloning a repository**

When you want to work with someone else's code, you first have to get it on your machine. For this task, use the **git clone** command. When you clone a repository, you copy files to a folder in the location where you run the command.

Use the following command to clone the correct sample code repository:

**git clone --branch main** [**https://github.com/CiscoDevNet/dne-devfun-code.git**](https://github.com/CiscoDevNet/dne-devfun-code.git)

From the first commit to the last, git clone copies locally to your computer all of the files that have ever been committed to the repository. When you first clone a repository, you see the default branch for a repository, which is usually called main.

Tip: To see the cloned repos in the in-browser file system, click the refresh icon until you see the directory.

From the repo root, you can see *all* the changes that were made in the history of the repo. Let's give it a try.

**git log**

If you use git log in a repo with many commits, press the spacebar to page through the commits and **q** to quit.

Preparing the Repo for Your Changes

You can verify which branch you are working on and the status of your working tree with the git status command:

**git status**

If you start editing files on the main branch and committing your changes, your local repository immediately goes out of sync with the remote server. That *might not* be a problem, but someone else can commit changes to the main branch and push their changes to the server before you push yours.

In that case, you have to **merge**, or reconcile, their changes with your own before you push your changes to the server. You cannot even pull updates from the server on the main branch until you reconcile the discrepancy.

Remember the story of the Git Guru at the beginning of this presentation? *"And how should I create a branch?" "Use git checkout."* Here it comes…

Run this command in the terminal to create a new branch called mycode:

**git checkout -b mycode**

Use the git checkout command to switch between branches. Adding the -b option instructs Git to create a new branch and then switch to it.

We created a new branch *and* we switched to it. That was the second thing or Git Guru prelude told us we would do with git checkout.

When you create a branch on your local repository, you are creating a **safe place** to edit files in the repo. Any changes that you make and commit are local to this new branch. You aren't making any commits to the branches that are synced with the remote server, so you can always pull down local copies of updates that other developers make to branches on the remote server.

Run the **git branch** command at any time to see which branches you have locally. The asterisk \* indicates which branch is active: *I would suggest that you think of branches in terms of what defines them: they’re a name for a particular commit and all the commits that are ancestors of it*

Keeping Your Local Repository Up-to-date

Over time, your local Git repository will get out of sync with the remote repository. As developers push commits to the server, you need to download these updates to your local repository. If you have been making your changes to your own local branch (which is the best practice), you can update your local repo with the updates from the remote repo with the git fetch command.

To download the latest updates, run the following command in the terminal:

**git fetch**

* Fetch only downloads updates but doesn’t integrate them. Git pull will download and integrate (merge or rebase).
* Git fetch, and then git merge, will download and integrate but give you greater insight into what is happening. Git pull can be smoother but comes at a cost – when something goes wrong (conflict), it’s harder to sort out exactly what happened.

You have cloned our repository, created a local branch for your edits, and know how to keep your local repo in sync with our remote. You are ready to make and commit your changes.

Changing files

If you change a file under Git's version control, Git knows. Open the intro-python/git-basics/change\_me.txt file in the in-browser editor and add, edit, or delete anything that you like. From the **git status** output, you can see that Git detected the modification to your files, and from the **git diff** output you can even see what changed in the files.

**Reverting changes**

Sometimes you make changes and then decide that you want to return to a past commit. Here are the simple workflows that can help you back out changes.

**Restore the last** committed **version**

Now comes the 3rd and final use of git checkout, as described by the Git Guru & the UNIX Programmer: *"And how should I update the contents of a single file in my working directory, without involving branches at all?"*

After you change a file, but before you commit it, you decide to revert those changes and restore the last committed version of the file back. Use git checkout to retrieve the last version of the file, overwrite the changes that you have made, and restore the file to its last committed version.

**git checkout** intro-python/git-basics/change\_me.txt

**Revert multiple changes**

After you change several files, you want to revert all those changes and restore the "last known good state": the last commit. Use git reset to restore your working directory to the last commit.

Note: You will lose all the changes that you have made since your last commit. *Be careful with this next command: make sure "git status" is clean, you're definitely on the correct branch*

**git reset –hard**

**Delete a branch**

After you create a branch to experiment with some changes, you decide that you want to discard the whole thing. Use git branch –delete to delete the branch.

**Git branch –delete –force <branch name>**

*Note: Always be careful when you see options like –hard or –force.* When you use these keywords, pause and think about what you are doing, because you lose some work when you run these commands. If that is your intention, proceed. Otherwise, think twice (or three times) before running these commands.

When you get your code working, you are going to want to save that and lock it away indelibly in the repository. To save the changes, you will “commit” your changes to the repository in the next step.

Making a Commit

Committing your changes to a repository is a two-step process:

* Add: *Stage* files to the commit with git add
* Commit: Commit the files to the repository with a git commit message and **git commit**

The -**m** option enables you to write a short commit message right there at the command line. Without the -m option, Git opens your environment's default text editor. Using a text editor lets you enter more details.

Make it your personal best practice to "commit often." Creating a commit is simple for you and a lightweight task for your computer. Committing often ensures that you capture more of the history of your code. You can go back and view commits and past changes anytime.

When you say to yourself, "Yeah! My code (or some portion of it) is working!" *then* commit.

A Sample Code Workflow

You now have a workflow whenever you perform the following tasks:

* Download (clone) a local copy of a remote repository to your workstation.
* Create a "safe place" (branch) for you to make edits.
* Save (commit) your incremental changes.

This macro workflow follows these steps:

1. Clone the Remote Repository. git clone <url>

2. Create and Checkout a Local Branch git checkout -b <new branch name>

3. Incrementally Commit Changes git add <new or modified file> git commit -m "Commit Message"

We can visualize a tree structure showing branches, commits and merges back into the main using a version of the git log command we saw before:

**git log --oneline --graph --decorate –all**

\* fc9371a Merge pull request #1 from CiscoDevNet/ag-py-folders

|\

| \* a7d70cf Remove rename the part1 and part2 folders on the main branch to match main-s

olutions

|/

| \* 8f876f2 (origin/main-solutions) Merge pull request #7 from pheilmei/fix-shebang\_ma

in\_solutions

| |\

| | \* 74f68cb adding # to shebang in types-loops-tools/structure.py

| |/

| \* e664f0d Update deck\_of\_cards.py

| \* 51f74c1 Update dive\_finds.py

| \* 083406f Adding main-solutions branch and code

|/

\* 0e399b6 Copying over files from meraki code

Creating a New GitHub repository

In a new tab in your browser, go to [https://github.com](https://github.com/) and log in.

Note: Create a free GitHub account if you have not already and generate a token. Refer to the Prerequisites in the Introduction for more information.

On your GitHub account page, create a new repository in your user space or organization named devnet-devfun. Use these settings:

In the terminal window, enter these git commands to add a "remote" named "origin" and then paste in the HTTPS reference, such as <https://github.com/justwriteclick/devnet-devfun.git>

**git remote add origin <paste the reference>**

In the terminal window, set the newly added remote "origin" as the upstream tracker, and push the initial commit to this new branch named main.

**git push --set-upstream origin main**

Note: Use your [authentication token](https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token) when prompted for a password. Go to the URL for the repository in GitHub and refresh the page. You should now see your README file in the default main branch. Well done!

* Managing Merge Conflicts

At some point, you're going to have a merge conflict. Conflicts arise when users have made overlapping changes to a file, and Git can't automatically merge the changes.

In this exercise, you will deliberately create a merge conflict and resolve it.

* Creating a merge conflict

**If you have any question, remember that you can contact me through the Cisco Webex App and I will be happy to converse with you there.**

Slide 17 Call to Action

Slide 18 Here are 3 other Git-related learning labs at Cisco DevNet

Introduction to Version Control Systems: Learn how to use Git for version control including configuration, commits, and differences.

Git Branching: Learn the basics of Git branching so that you can diverge from the main code base.

Using Git with servers: Learn how to use Git with servers to share your work with others.

Slide 19 The New learning labs at DevNet contain **29 learning tracks**, with **116 learning modules**, with **451 learning labs**, broken down into the categories of **Collaboration, Cloud, Networking, Analytics & Automation, IoT, Data Center, Security, and Observability**

Slide 20 Don’t forget to fill out your session surveys

Attendees who fill out a minimum of four session surveys and the overall event survey will get Cisco Live-branded socks (while supplies last)!

Attendees will also earn 100 points in the Cisco Live Game for every survey completed.

These points help you get on the leaderboard and increase your chances of winning daily and grand prizes

Slide 21

* Visit the Cisco Showcase for related demos
* Book your one-on-one Meet the Engineer meeting
* Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
* Visit the On-Demand Library for more sessions at [www.CiscoLive.com/on-demand](http://www.ciscolive.com/on-demand)

Slide 22 Thank you