Up and Running with Git (v3)

Author: James D. Triveri
Date: 2021-03-11

Covered Today:

- Collaborative development overview
- Introduction to Git and basic commands
- Creating a dedicated repository folder locally

- **Team exercise:**
  - Cloning an existing source code repository
  - Creating a development branch
  - Updating working copy with new content
  - Staging and committing changes locally
  - Pushing changes to shared central repository.

Covered Next Time:

- Creating new project repositories
- Additional basic and intermediate Git commands
- Basic conflict resolution
- Reverting to previous commits
- Customizing `.bashrc`

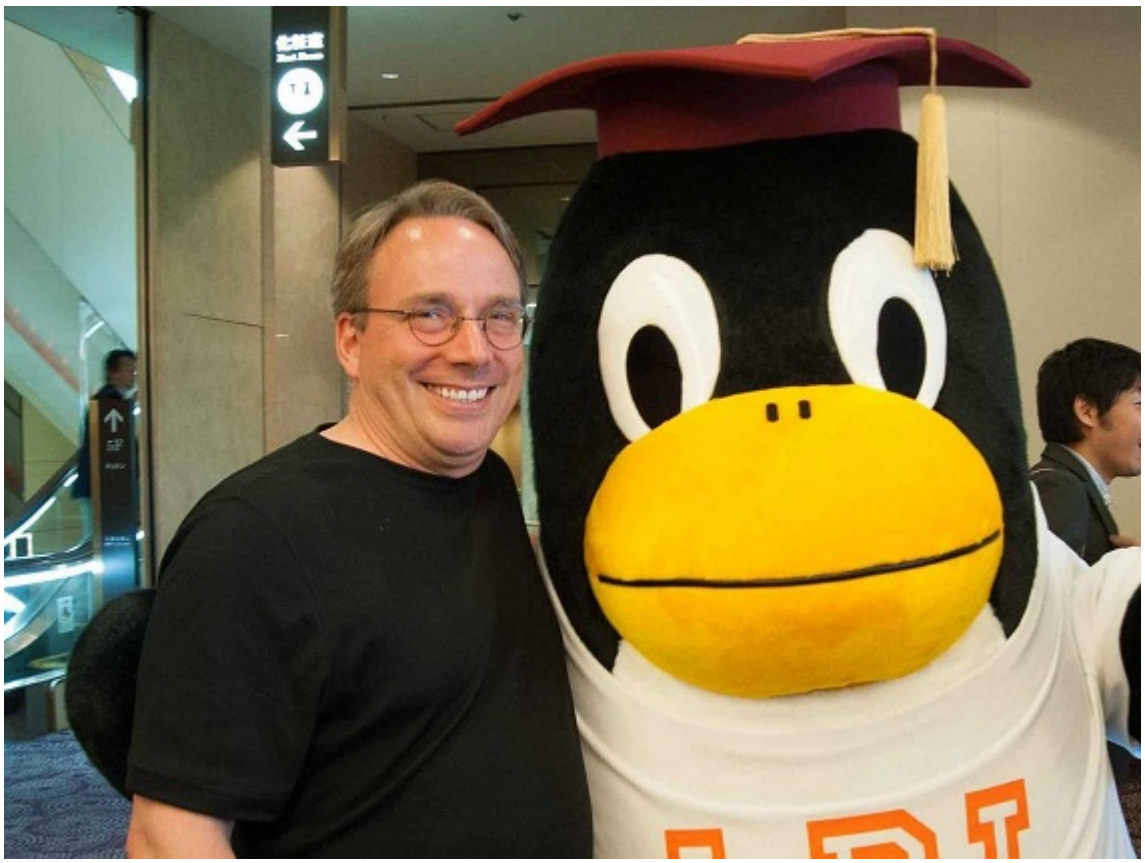
Very Brief Git Overview:

- https://www.youtube.com/watch?v=hwP7WQkmECE

Git is the canonical example of a Distributed Version Control System. Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion, in Git, every developer's working copy of the code is also a repository that can contain the full history of all project changes. In addition to being distributed, Git has been designed with performance, security and flexibility in mind.

Difference Between GitHub and Git?

- **Git** is the preferred software application that handles version control. Over the past few years, it has become the de facto standard for code management and versioning. Created by Linus Tsorvalds, the creator and maintainer of Linux.



- **GitHub** is a platform that serves as a central location which facilitates collaborative development. Similar platforms include GitLab and BitBucket (which is used at GuideOne). It relies on Git for source code management.

- Easier to get started today than 5-10 years ago: No longer relevant competing systems like cvs, svn, hg, mercurial, etc.

Changing our Approach to Collaborative Development

- Transition from everyone working on a single set of source files, to to each individual working/developing/testing code and changes within a personal copy of the project source files (referred to as a "working copy").

- Tensorflow example: https://github.com/tensorflow/tensorflow (https://github.com/tensorflow/tensorflow)

- Changes/additions/revisions are first performed in user's working copy. Once modifications are deemed sufficiently error free, changes are synced with the project's shared central repository on Bitbucket.

Group Exercise: Demo Project

Part I: Creating a Working Copy

Let's create a dedicated `Repos` folder to hold working copies for all the projects you work on. Be sure to create it on `C:/` . You will experience severe performance degradation if you store your working copies to the shared file server. A good location is:

```
C:/Users/[ID]/Repos
```

In what follows, replace `i103455` with your ID...

[0.] Open the Git application.

[1.] Navigate to `C:/Users/i103455` using the `cd` command:

```
$ cd C:/Users/i103455
```

[2.] Create a `Repos` directory using `mkdir` , then use `cd` to switch into `Repos` :

```
$ mkdir Repos
$ cd Repos
```

[3.] Updade your username and email in Git's global settings:

```
$ git config --global user.name "Triveri, James"
$ git config --global user.email "JTriveri@guideone.com"
```

[3.] Clone the *Demo* project on Bitbucket (everyone should have access). This will create a `Demo` folder within your newly created `Repos` directory. This "cloned" repository is your working copy of the `Demo` project source files.

```
$ git clone ssh://git@git.guidehome.com:7999/ac/demo.git
```

> Note: If you didn't setup ssh authentication using the guide I sent out a while back, you'll need to replace `ssh://git@git.guidehome.com:7999/ac/demo.git` with `https://git.guidehome.com/scm/ac/demo.git`

[4.] Switch into the `Demo` directory. Create a new branch named `dev`:

```
$ cd Demo
$ git checkout -b dev
```

Useful Keyboard Shortcuts when working from the shell:

- `CTRL + A` : Move cursor to beginning of line.
- `CTRL + E` : Move cursor to end of line.
- `CTRL + U` : Clear line.
- `CTRL + INSERT` : Copy highlighted text from shell.
- `SHIFT + INSERT` : Paste clipboard contents into shell.
- `CTRL + L` : Clear screen.

Part II: Modifying, Pushing and Pulling Changes

The repository's master branch is used for syncing changes between your working copy and the shared central repository.

Git version control mantra:

> **Never commit changes directly to master!!!**

Before we start, from the R console in RStudio, set the current working directory to the top-level of the cloned Demo repository as follows:

```
> setwd("C:/Users/i103455/Demo")
```

Each individual will share their screen and execute the following commands.

[0.] Add a new file to the Demo project named using your initials (do this from RStudio). For example, Kate would create a file named `KS.R`.

[1.] In the file, add a line similar to what is found in `JDT.R`, but replacing my name with yours. For example, `KS.R` would contain the following:

```
# Contents of KS.R in Demo project.

message("Kate Sajewich has contributed to Project Demo!")
```

[2.] Stage and commit changes locally to the dev branch. You must include a message each time a commit is created. Each time a commit is made, a restore point to the current state of the project is created. It's a good habit to commit often. In what follows, we first check to make sure we're committing to the dev branch, not the master branch:

```
$ git branch
* dev
  master
```

Stage and commit changes on dev branch:

```
$ git add --all                    # stage local changes
$ git commit -m "JDT: Added JDT.R." # commit changes to local dev bra
nch.
```

[3.] Publish our changes to the shared central repository ("origin") so other team members can integrate your changes. Changes are integrated into the shared central repository via the master branch, and is carried out in 4 steps:

First, switch to the master branch:

```
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```

Second, pull changes down into local master branch from shared central repository. **If you've made any changes since checking out master, they will be overwritten!**

```
$ git pull
```

Third, merge the changes committed to dev branch with master branch:

```
$ git merge dev
```

Fourth, push merged master branch up to shared central repository:

```
$ git push -u origin master
```

Once you've published your commits to the shared central repository, be sure to checkout your local dev branch again so you don't inadvertently make changes to your local master branch.

> **Never commit changes directly to master!!!**

Why?

Demonstration

Cloning actenv

To create a working copy of the reserving environment code, we proceed as above:

[0.] Navigate to your `Repos` directory.

```
$ cd Repos
```

[1.] Clone actenv from Bitbucket:

```
$ git clone ssh://git@git.guidehome.com:7999/ac/actenv.git
```

[2.] Switch into `actenv` folder and checkout a local dev branch:

```
$ cd actenv
$ git checkout -b dev
```

You can get a summary of the changes made to the project thus far by running the `git log` command:

```
$ git log --oneline
```

This will print a list of commits + commit messages, each confined to a single line.

Now you can play with the source files without fear of breaking anything. If you do happen to break something, simply delete the working copy and create a new one. Do this as often as necessary. This is one of the major advantages of maintaining code with version control.

- Merge conflicts: https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-merge-conflicts (https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-merge-conflicts)

- Resolving merge conflicts: https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/resolving-a-merge-conflict-using-the-command-line (https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/resolving-a-merge-conflict-using-the-command-line)