Introduction to Git and Github

Software Tools for Business Analytics: Lecture 1

Galit Lukin

Based on Slides By Jackie Baek

MIT

January 14th, 2019

What is Git and GitHub?

- **Git** is an open source, *distributed version control system*.
  - Other version control systems include mercurial, svn, perforce.
  - Git is modern (2005) and most popular.

What is a version control?

- The management of changes to documents (in a codebase)
- Can be used for projects big or small, long-term or short-term.

What is a distributed version control system?

- Software that stores "snapshots" of a project over time.
- These "snapshots" are mirrored every developer's computer

What is a distributed version control system?

- Software that stores "snapshots" of a project over time.
- These "snapshots" are mirrored every developer's computer
- Every developer has the full history of the codebase mirrored on their computer

What is GitHub?

- **GitHub** is a service that allows you to host projects using Git.
- **Git** is a command-line tool
- **Github** is where developers store their projects as Git repositories

Why should I learn Git?

- ▶ Everyone uses it.
  - ▶ We'll be using it in this class.
- ▶ Backup (in the cloud).
- ▶ Versioning with fine granularity.
- ▶ Collaboration.
  - ▶ But useful even when working by yourself.

Why should I learn Git?

- ▶ Everyone uses it.
    - ▶ We'll be using it in this class.
- ▶ Backup (in the cloud).
- ▶ Versioning with fine granularity.
- ▶ Collaboration.
    - ▶ But useful even when working by yourself.

Can't we just use Dropbox?

Why should I learn Git?

- ▶ Everyone uses it.
    - ▶ We'll be using it in this class.
- ▶ Backup (in the cloud).
- ▶ Versioning with fine granularity.
- ▶ Collaboration.
    - ▶ But useful even when working by yourself.

Can't we just use Dropbox?

- ▶ Git gives finer granularity: files vs. lines within a file.
- ▶ This granularity is essential when writing code.
- ▶ Easy to:
    - ▶ share code
    - ▶ merge code
    - ▶ retract changes
    - ▶ look at the full history of the code
    - ▶ work in an organized way as an individual and as a group

Terminology

- **repository (repo):** the project that contains all files.

Terminology

- **repository (repo):** the project that contains all files.
- **commit:**
  - As a noun: one snapshot of the repository, a single point in the Git history of the repo
  - As a verb: The action of storing a new snapshot of the project's state in the Git history

Terminology

- **repository (repo):** the project that contains all files.
- **commit:**
  - As a noun: one snapshot of the repository, a single point in the Git history of the repo
  - As a verb: The action of storing a new snapshot of the project's state in the Git history
- **branch:** an active line of development.
  - a single Git repository can track an arbitrary number of branches.

Terminology

- **repository (repo):** the project that contains all files.
- **commit:**
  - As a noun: one snapshot of the repository, a single point in the Git history of the repo
  - As a verb: The action of storing a new snapshot of the project's state in the Git history
- **branch:** an active line of development.
  - a single Git repository can track an arbitrary number of branches.
- **checking out** a branch: updates the files in the working directory to match the version stored in that branch. A way to select which line of development you're working on.
- **HEAD:** the current branch.

Terminology

- **repository (repo):** the project that contains all files.
- **commit:** one snapshot of the repository.
- **branch:** an active line of development.
- **checking out** a branch: A way to select which line of development you're working on.
- **HEAD:** the current branch.
- **local:** repository sitting on your local machine.
- **remote:** repository sitting on a remote machine (e.g. GitHub).

Terminology

- **repository (repo):** the project that contains all files.
- **commit:** one snapshot of the repository.
- **branch:** an active line of development.
- **checking out** a branch: A way to select which line of development you're working on.
- **HEAD:** the current branch.
- **local:** repository sitting on your local machine.
- **remote:** repository sitting on a remote machine (e.g. GitHub).
- **pull:** grab changes from remote (or other branch) to local.
    - **fetch:** downloads commits and files from a remote repository into your local repo
    - **merge:** take two lines of development and integrates them into a single branch
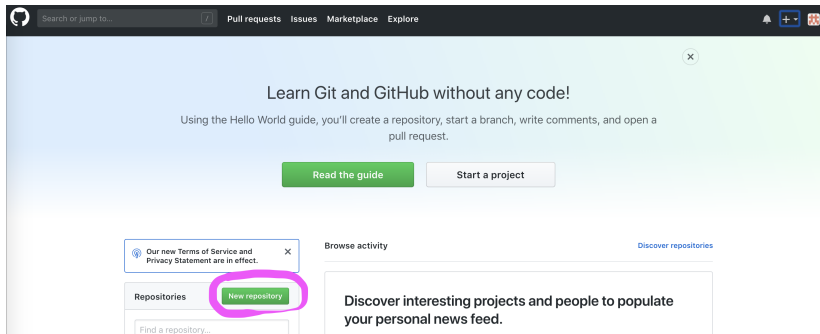   - pull == fetch+merge
- **push:** update remote with local changes.

Typical Workflow
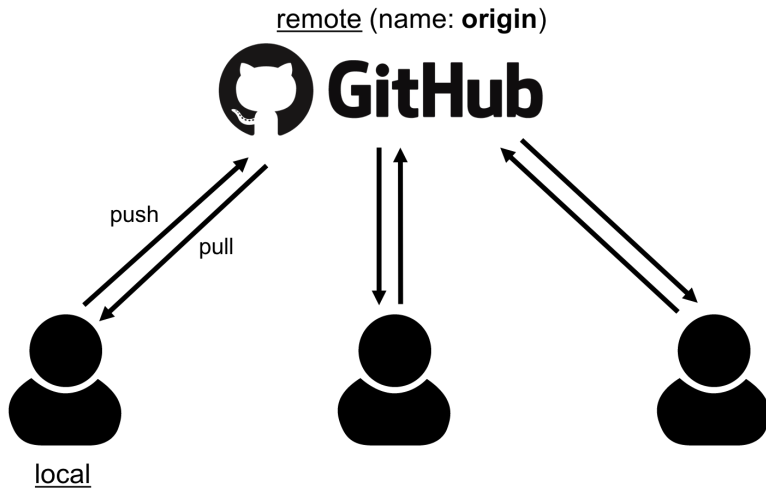
Typical Workflow
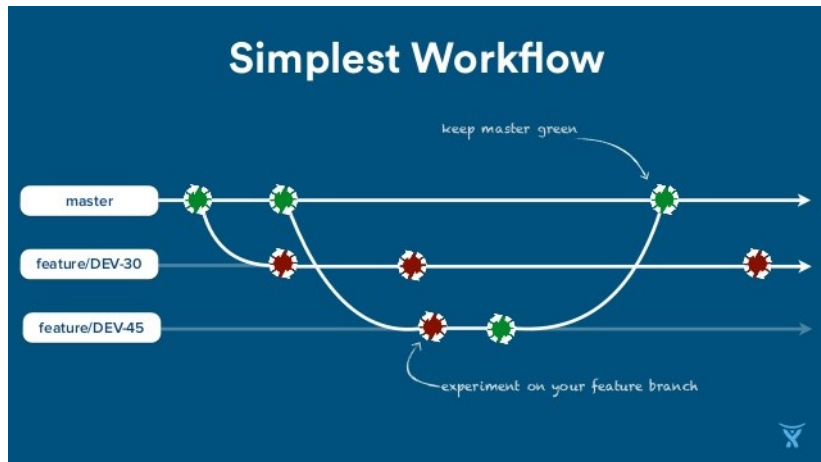
Creating a new repository

Creating a new repository

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner   **Repository name**

🐙 galitLukin ▾ / playground ✓

Great repository names are short and memorable. Need inspiration? How about **cautious-lamp**.

Description (optional)

playground for Computing in Optimization and Statistics

⦿ 🖥 **Public**
     Anyone can see this repository. You choose who can commit.

○ 🔒 **Private**
     You choose who can see and commit to this repository.

☑ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾ | Add a license: **None** ▾ ⓘ

Create repository

Creating a new repository

Cloning a repository

$ git clone <URL>

- ▶ Go to any repository and copy the URL
- ▶ This will create a new directory with the same name as the repository name and clone the repo there.

```
$ git clone https://github.com/galitLukin/playground
```

Cloning a repository

$ git clone <URL>

▶ Go to any repository and copy the URL
▶ This will create a new directory with the same name as the repository name and clone the repo there.

```
$ git clone https://github.com/galitLukin/playground
```

```
$ cd playground
$ git status
$ git config core.editor "nano"
```

Let's make some changes

▶ Create a new file called new_file.txt
  ▶ Add "This is a new file"
▶ Modify existing_file.txt
  ▶ interesting → uninteresting

Let's make some changes

- Create a new file called new_file.txt
  - Add "This is a new file"
- Modify existing_file.txt
  - interesting → uninteresting

```
$ cd playground
$ nano new_file.txt
   This is a new file
$ nano existing_file.txt
   interesting -> uninteresting
```

See what changed and your branch's status

```
$ git diff
```

▶ Shows what changed since the last commit

See what changed and your branch's status

```
$ git diff
```

▶ Shows what changed since the last commit

```
$ git status
```

▶ See which changes have been staged, which haven't, and which files aren't being tracked by Git

See what changed and your branch's status

```
$ git diff
```

▶ Shows what changed since the last commit

```
$ git status
```

▶ See which changes have been staged, which haven't, and
which files aren't being tracked by Git

```
$ git log
```

▶ Lists the commits made in that repository in reverse
chronological order

File states

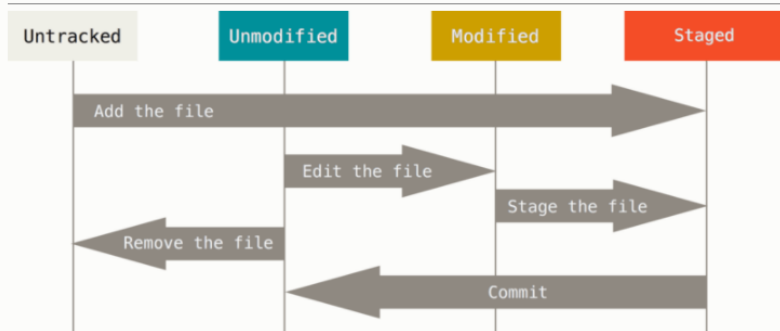- Git will notice any file in the directory of the repository.

File states

- Git will notice any file in the directory of the repository.
- A file is either **untracked** or **tracked**.

File states

- ▶ Git will notice any file in the directory of the repository.
- ▶ A file is either **untracked** or **tracked**.
- ▶ A tracked file may be:
    1. **Unmodified:** No changes since the last commit.
    2. **Modified:** Changes have been made to it since the last commit.
    3. **Staged:** Changes will be committed in the next commit.

File states

- Git will notice any file in the directory of the repository.
- A file is either **untracked** or **tracked**.
- A tracked file may be:
  1. **Unmodified:** No changes since the last commit.
  2. **Modified:** Changes have been made to it since the last commit.
  3. **Staged:** Changes will be committed in the next commit.

Staging files

```
$ git add <filepath>
```

- Any *untracked* or *modified* file that is added will be *staged*.
- Each such file will be included in the next commit.

Staging files

<center>

`$ git add <filepath>`

</center>

- Any *untracked* or *modified* file that is added will be *staged*.
- Each such file will be included in the next commit.

```
$ git add new_file.txt
$ git add existing_file.txt
```

Use Git add to either:
- Add a new file to the repository (untracked → staged)
- Record a change that you made to an existing file (modified → staged)

Git commit

```
$ git commit -m <commit message>
```

▶ This creates a new snapshot of our repository with all changes that we have staged.

Git commit

$ git commit -m <commit message>

▶ This creates a new snapshot of our repository with all changes that we have staged.

```
$ git commit -m "Added file and modified existing."
```

▶ This new snapshot (commit) is saved in our local repository.

▶ This *does not* push our changes to the remote repository (GitHub).

Git commit

```
$ git commit -a -m "Added file and modified existing."
```

- ▶ The *-a* tells the command to automatically stage files that have been modified and deleted
- ▶ New files that Git has not recognized are not affected.

```
$ git add .
$ git commit -a -m "Added file and modified existing."
```

Interacting with remote

$ git push

- Update remote repository with local commits.

$ git pull

- Updates local repository with remote commits.

Merging

- When we 'git pull', Git fetches the remote repository from GitHub and *merges* the new remote updates with our local repository.
- Even if both remote and local modified the same file, Git is *usually* able to correctly merge the two copies.
- We get a **merge conflict** if both parties modified the *same parts of the same file*.

Merging

```
$ git pull
```

```
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/galitLukin/playground
   8bfcca0..b3207b5  master     -> origin/master
Auto-merging existing_file.txt
CONFLICT (content): Merge conflict in existing_file.txt
Automatic merge failed; fix conflicts and then commit the result
```

Resolving Merge Conflicts

```
$ cat existing_file.txt
```

Resolving Merge Conflicts

```
$ cat existing_file.txt

<<<<<<< HEAD
Did a different first edit!
=======
Did the first edit!
>>>>>>> b3207b5d9cddd22934ccd2fed0a6cc16eefdab73
```

- The markers $<<<<<<<$, $=======$, $>>>>>>>$ indicate the conflict.
- The section in between the first two markers is your local change (HEAD), while the bottom section indicates the update from remote.
- Must resolve conflict manually by editing the file, making sure to get rid of the conflict markers.

Resolving Merge Conflicts

```
$ cat existing_file.txt

<<<<<<< HEAD
Did a different first edit!
=======
Did the first edit!
>>>>>>> b3207b5d9cddd22934ccd2fed0a6cc16eefdab73
```

- The markers <<<<<<<, =======, >>>>>>> indicate the conflict.
- The section in between the first two markers is your local change (HEAD), while the bottom section indicates the update from remote.
- Must resolve conflict manually by editing the file, making sure to get rid of the conflict markers.

```
$ nano existing_file.txt
```

Resolving Merge Conflicts

▶ After resolving conflicts, we must add the file for staging and commit again.

▶ Git will automatically create a commit message: "Merge branch 'master' of https://github.com/galitLukin/ playground"

Resolving Merge Conflicts

- ▶ After resolving conflicts, we must add the file for staging and commit again.
- ▶ Git will automatically create a commit message: "Merge branch 'master' of https://github.com/galitLukin/playground"

```
$ git add existing_file.txt
$ git commit
```

- ▶ At this point, we can push.

Typical Workflow

```
Fetch remote changes.
  $ git pull

(If there are any conflicts, resolve them and commit.
  $ git add <conflicted files>
  $ git commit)

Make changes. Stage modified and new files.
  $ git add <files>

Commit changes.
  $ git commit -m "this is my commit message"

(If editing took a while...
  $ git pull
  And if needed, resolve merge conflicts)

Push local changes to remote.
  $ git push
```
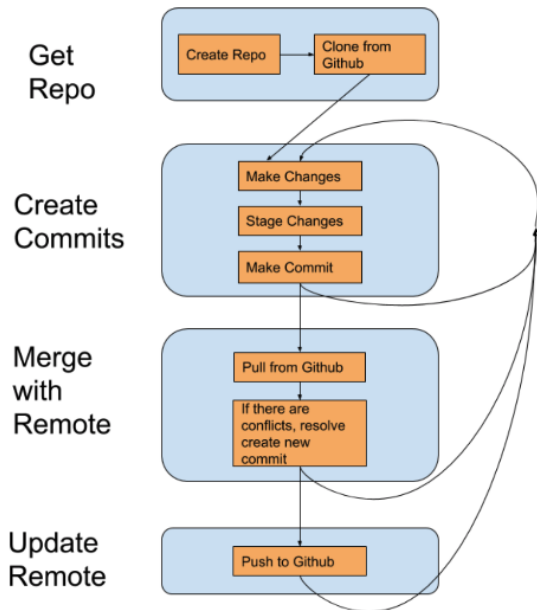
Typical Overall Workflow

Useful tips

- Google is your friend. (e.g. "How to undo merge in Git".)
- Almost anything can be undone, as long as it is committed.
- Commit often, pull often.
- Might take a while to get used to, but is useful knowledge that will improve productivity and collaboration.

Thank you!