



Version Control 101

> Your guide to mastering Git, using Github, and mastering version control via an interactive tutorial

Version Control? Why?

- “The task of keeping a software system consisting of many versions and configurations well organized.”
 - ie. Stop creating a new project every time your code stops working!
 - Never lose a file and easily revert to a previous working state
- Why?
 - Because you’ll **absolutely** be using it in your future job...
 - Employers often look at your Github for previous projects/experience
- Available Version Control Tools
 - Revision Control System (RCS) - comes setup on a linux machine
 - Subversion, CVS, Mercurial, **GIT!!!**

Learn some Lingo!

- **Repository (Repo)** : The database storing the files
- **Server**: The computer storing the repo
- **Client**: The computer connecting to the repo (often your local machine)
- **Working Set/Working Copy**: Your local directories of files (where you make changes)
- **Trunk/Main**: The primary location for code in the repo - ie. the main line

Learn some (more) Lingo!

- **Add:** To put a file into the repository for the first time (being to track it with version control)
- **Revision:** What version a file is on
- **Head:** The latest revision in the repository
- **Check out:** Download a file from the repository
- **Check in:** Upload a file to the repository (if it has changed)
- **Check in Message:** A short message made to a file since it was created
- **Changelog:** A list of changes made to a file since it was created
- **Sync:** Synchronize your Files with the latest from the repository.
- **Revert:** Throw away your local changes and reload the latest version from repo

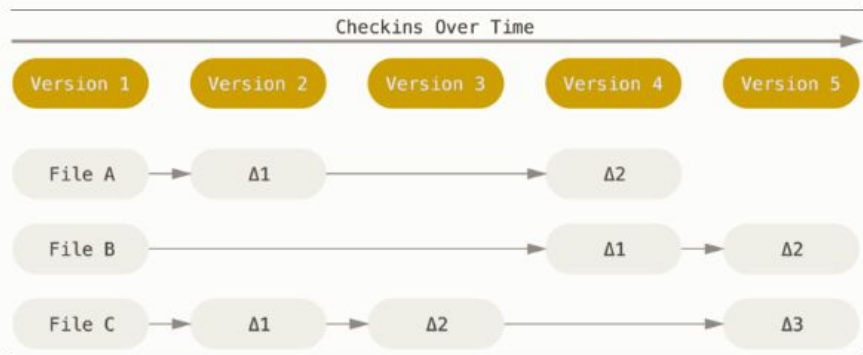
Git Basics 101

- Simply a distributed version control system tool
 - I.e. it allows for a team to work together on the same set of files at the same time.
 - Is also used by software teams to fix bugs in their projects and release updated revisions
- How does it work?
 - Git needs to be installed on your client (often your local machine) in order to be run -
 - For Linux Users:
 - Binary installer: `> sudo yum install git-all`
 - Ubuntu: `> sudo apt-get install git-all`
 - For Mac Users:
 - Git Installer (<http://git-scm.com/download/mac>)
 - For Windows Users:
 - Git Installer (<http://git-scm.com/download/win>)

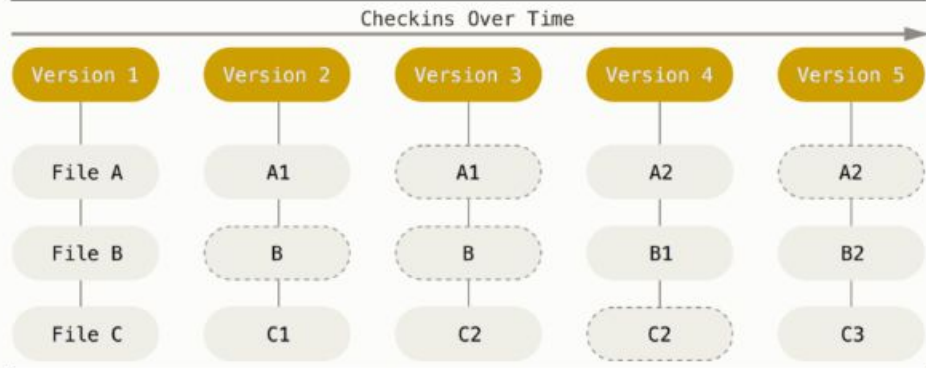
Git Basics 101 (Cont.)

- Git vs. other standard version control systems
 - Files vs. Snapshots - Standard version control systems keep track of the changes in a file with every iteration whereas Git thinks of its data more like a “stream of snapshots”

Standard Version Control



Git



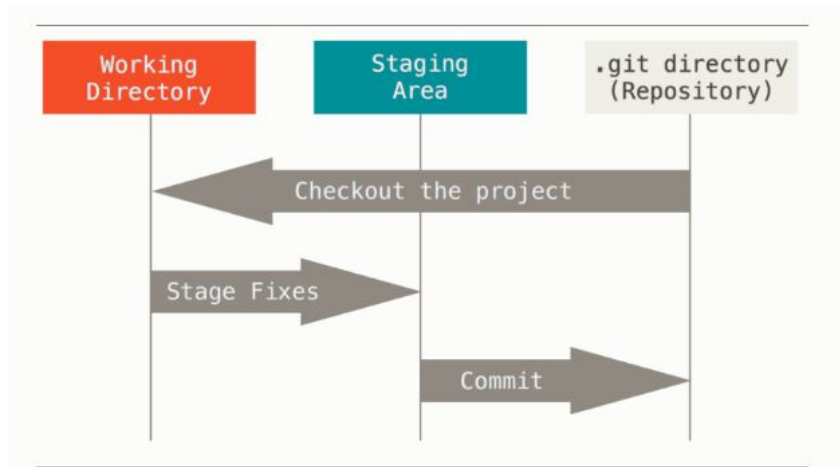
Git: The basic idea

MAIN THING TO REMEMBER: Git has 3 main stages your files can reside in - committed, modified, and staged.

- Committed: Data is safely stored in your local database
- Modified: Means that you have changed the file but not committed yet
- Staged: Marked a modified file in its current version to go into next commit snapshot

Basic Workflow:

1. You modify files in your working directory
2. You stage files
3. You commit which takes the stages files and stores snapshot



Let's "Git" Started

- Download and Install GIT onto your computers
 - To check if GIT installed properly, type “git --version” in your terminal and if installed, it will print the git version number that is currently installed
- It's a good idea to introduce yourself to Git with your name and public email address before doing any operations
 - To accomplish this, we use the command: **git config**
 - > git config --global user.name “YOUR NAME GOES HERE”
 - > git config --global user.email “you@yourdomain.com”
 - To see your username and email, you can also do the following:
 - > git config user.name
 - Or, you can see all the set config variables via: **git config --list**

Let's “Git” started with a Project

There are a few options that are available when starting a project:

- Start a project from scratch
 - Make a new directory that you would like to dedicate to the project
 - > **git init** // Used to initialize the working directory with git
 - Git will reply with: “Initialized empty Git repository in .git/”
- Copy a project from a remote server
 - Done via the **clone** command
 - > **git clone username@host:/path/to/repository**
 - > **git pull** // Pull the latest version of the files from master trunk
 - Do the following right now:
 - `git clone https://github.com/PreritO/HKN_Git_Tutorial_FA16.git`

Adding/Committing files

- To add files to the staging area, we use the following command: **git add**
 - To add all of the files to the staging area
 - `> git add *`
 - To add specific files to the staging area
 - `> git add <filename1> <filename2> <filename3>`
- To commit files to repository
 - Automatically commits all the files in the staging area (ie. all the files that you “added”)
 - Done with the following command “**git commit**”
 - To commit files with a message
 - `> git commit -m “YOUR MESSAGE GOES HERE”`
 - Instead of using git add over and over again, you can also use: “**git commit -a**” which automatically notice any modified (but not new) files, add them to index and commit in 1 step

Pushing Local Changes to Remote Repo

After you have “committed” your changes, they are now in HEAD of your LOCAL working copy. To push those changes to your remote repo, execute the following:

- **git push origin <branch name>**
 - To push to master branch - **git push origin master**

You can also see the differences between your local files and the remote files via:

- **git diff --cached**
 - Without --cached option, git will show you any changes that you've made but not added to the index

File Status

Often, between adding, committing and pushing files, it can be hard to keep track of all the files in the repository. As a result, git makes it easy for us to view the status of our files.

- **git status**

```
$ git status
On branch master
Changes to be committed:
  Your branch is up-to-date with 'origin/master'.
  (use "git reset HEAD <file>..." to unstage)

        modified:   file1
        modified:   file2
        modified:   file3
```

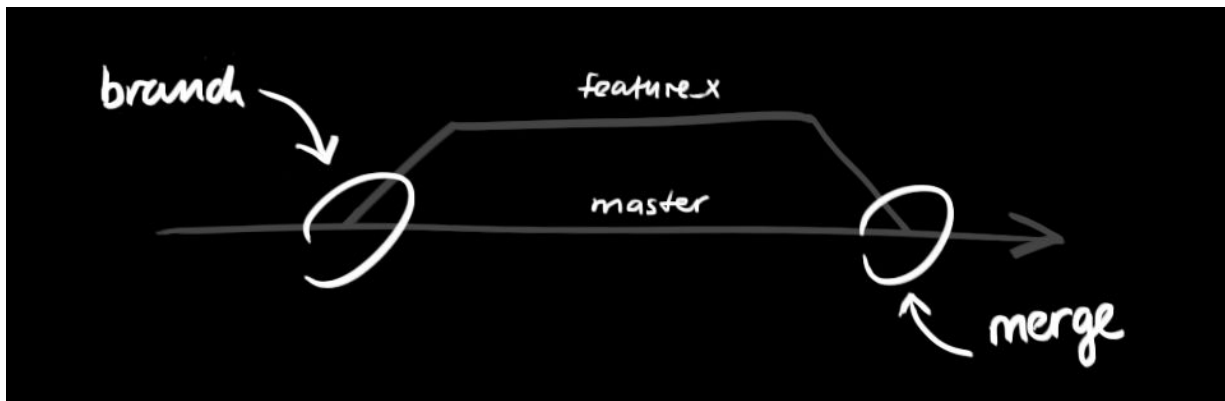
We can also track the log of our commits via: **git log**

- To see complete diffs at each step: **git log -p**
- To see overview of the change in each step: **git log --stat --summary**

Branching

Branching is probably one of the most useful tools of Git!

- The **master** branch is the default branch when you create repo, and so in order to add various feature to our main application, we can branch out repo



Branching (cont.)

- Thus, in order to add a new feature to your application without changing all the code, we can just create a new branch and then “merge” this branch with our master to incorporate our updates
 - To create a new branch: **git checkout -b <branch name>**
 - le. > git checkout -b feature_x
 - To switch branches: **git checkout <branch name>**
 - To switch to master branch: > git checkout master
 - To delete a branch: **git branch -d <branch name>**
 - le. > git branch -d feature_x
 - To simply view all your branches: **git branch** (Note: The * marks the current branch)
- Note: A Branch is **NOT AVAILABLE TO OTHERS** unless you push the remote repository branch
 - To push the remote repo branch: **git push origin <branch>**

Branching (cont.) - Exercise

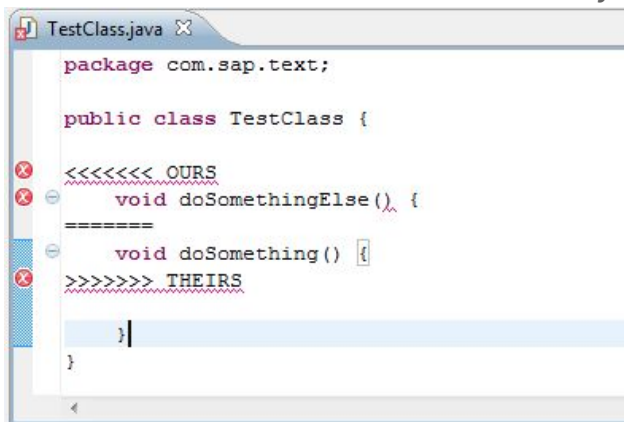
- Now, create a new branch with the name: <lastname_firstname_branch>
- Modify file.txt to add your name, email address, and a fun fact about you!

```
First Name:  
Last Name:  
Email Address:  
Fun Fact:|
```

- Add, Commit, and Push this file back to YOUR branch! NOT the Master branch (refer to previous slide on how to do this)
- When finished, feel free to help others around you!

Update and Merging Local Branch

- To update your local directory with the newest commit from the remote repository, we use the command: **git pull**
- To merge another branch in your active branch, use the command: **git merge <branch>**
 - Git tries to auto-merge changes but this doesn't always work... instead, it lets you know which files have **conflicts** in them and the conflicts must manually be resolved



The screenshot shows a code editor window titled 'TestClass.java'. The code contains a package declaration, a class declaration, and two methods. A conflict is shown between two versions of the 'doSomethingElse()' method. The editor uses a visual representation of the conflict: the original code is shown in a light blue background, and the conflicting code is shown in a light red background. The conflict is marked with red 'X' icons on the left margin. The code is as follows:

```
package com.sap.text;

public class TestClass {

    <<<<<<< OURS
    void doSomethingElse() {
        =====
    void doSomething() {
    >>>>>>> THEIRS

    }
}
```


Merging (Cont.)

To manually fix the conflicts, remove the arrows “<<” and the equal signs “==” and then keep the code that you want to push (note, your code will be before the equals sign)

- Once you’ve decided which code to keep, you need to mark them as “merged” with the following git command: **git add <filename>**
- You can also see the changes in your file in the different branched by: **git diff <source_branch> <target_branch>**

Replacing Local Changes

Sometimes, you might want to revert to the remote repo version of the file and overwrite all your local changes. To do this:

- We execute the following set of commands:
 - > **git fetch origin**
 - > **git reset --hard origin/master**

Merging - Exercise

Now, as an exercise to understand merging, complete the following steps:

- Create another branch with the name <lastname_firstname_branch_merge>
- Edit file.txt by adding a DIFFERENT fun fact about yourself
- Add, commit, and push your new file.txt to you NEW MERGED branch
- Go back to your original branch (NOT master branch) and merge your new branch with the old branch. What happens? How do you fix this? Try typing “git status” and see that prints on the screen.
- Once you’ve edited the files to resolve the conflicts, you can commit the new file to your original branch
- Congrats, you’re done!

Questions?

Additional Resources

- <https://try.github.io/levels/1/challenges/1>
- <https://git-scm.com/docs/gittutorial>
- <https://www.atlassian.com/git/tutorials/>
- <http://www.vogella.com/tutorials/Git/article.html>
- <https://www.kernel.org/pub/software/scm/git/docs/gittutorial.html>
- <https://www.codecademy.com/learn/learn-git>
- <http://learngitbranching.js.org/>
- <https://blog.udemy.com/git-tutorial-a-comprehensive-guide/>