Everyday GIT

* Introduction

For everyone who is used to SVN, GIT might be a bit confusing and complex at first, however once you understand and get used to it, then it’s the simplest content manager.

* GIT vs Subversion(e.g SVN)

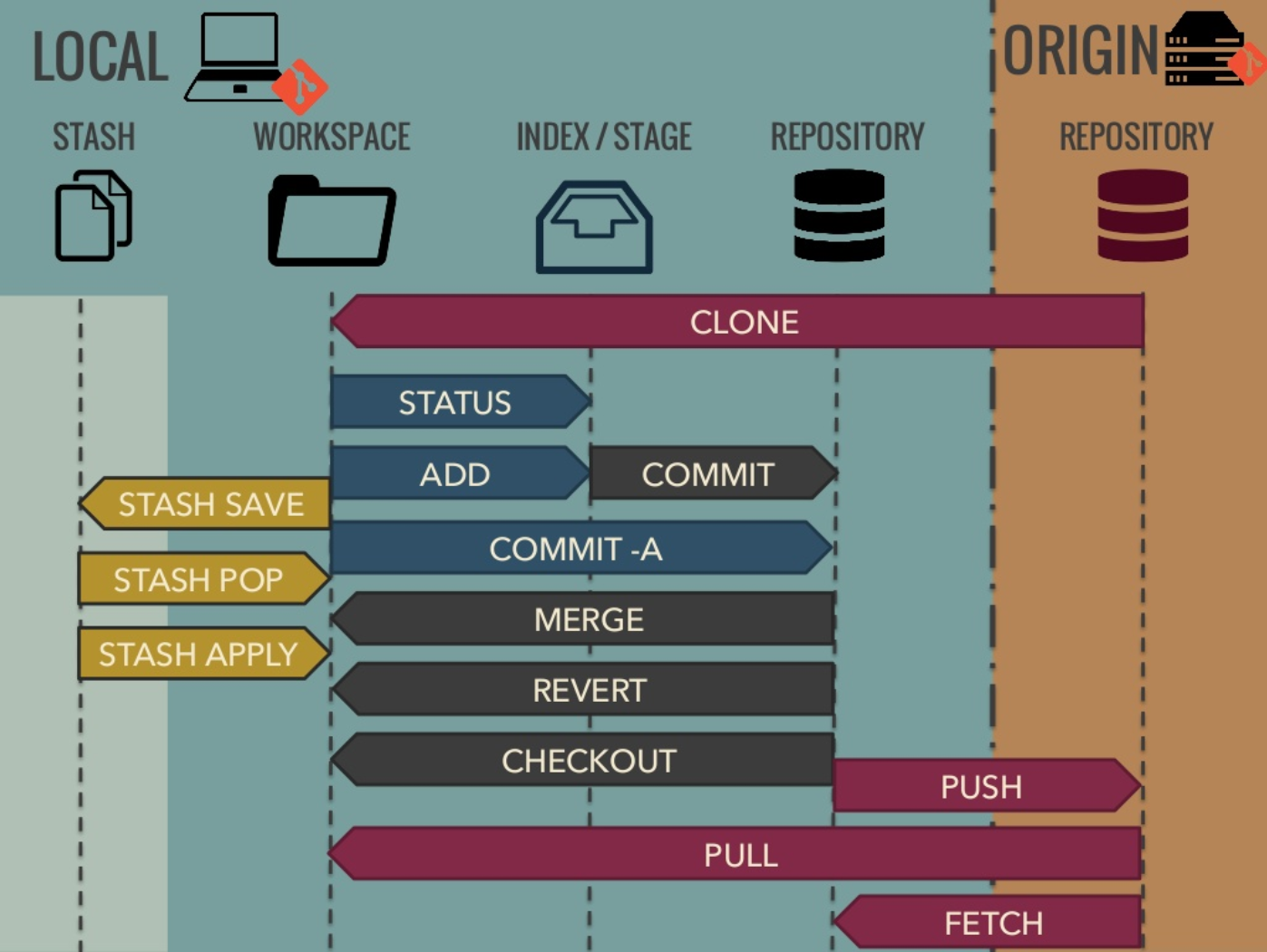
The key difference is that it is decentralized. Imagine you are a developer and you develop on your laptop and you want to have source control so that you can go back few hours.

With Subversion, you have a Problem: The SVN Repository may be in a location you can't reach (in your company, and you don't have internet at the moment), you cannot commit. If you want to make a copy of your code, you have to literally copy/paste it.

With Git, you do not have this problem. Your local copy is a repository, and you can commit to it and get all benefits of source control. When you regain connectivity to the main repository, you can commit against it.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** |  | **Git** | **SVN** |
| Commit Files (A) | Add, commit and push 113 modified files (2164+, 2259-) | 0.64 | 2.60 | 4x |
| Commit Images (B) | Add, commit and push 1000 1k images | 1.53 | 24.70 | 16x |
| Diff Current | Diff 187 changed files (1664+, 4859-) against last commit | 0.25 | 1.09 | 4x |
| Diff Recent | Diff against 4 commits back (269 changed/3609+,6898-) | 0.25 | 3.99 | 16x |
| Diff Tags | Diff two tags against each other (v1.9.1.0/v1.9.3.0 ) | 1.17 | 83.57 | 71x |
| Log (50) | Log of the last 50 commits (19k of output) | 0.01 | 0.38 | 31x |
| Log (All) | Log of all commits (26,056 commits - 9.4M of output) | 0.52 | 169.20 | 325x |
| Log (File) | Log of the history of a single file (array.c - 483 revs) | 0.60 | 82.84 | 138x |
| Update | Pull of Commit A scenario (113 files changed, 2164+, 2259-) | 0.90 | 2.82 | 3x |
| Blame | Line annotation of a single file (array.c) | 1.91 | 3.04 |  |
| Clone | Clone and shallow clone(\*) in Git vs checkout in SVN | 107.5 | 14.0 |  |
| Size | Size of total client side data and files after clone/checkout (in M) | 181.0 | 132.0 |  |

* Git Workflow



From right to left:

- **Origin Repository** the actual repository residing on server.

- **Repository** the local copy of the whole ‘Origin Repository’.

- **Index** (or Staging Area) is the place where changes are collected for a commit. It is like listing of a changes(files which has modified/ added) you wish to commit.

- **Workspace** which is just the editable files in the normal filesystem (like Working copy in SVN)

- **Stash** is an extra stack for storing backups of changes ( this allows to remove changes from Workspace without loosing them)  
... there are only a few cases where you have to use the Stash, e.g.

1. When pulling changes that conflict with your workspace.
2. Suddenly working on some different task, may be different branch. So you stash your changes (generally incomplete which cannot be committed), finish other branch work. Come to earlier branch apply the stash and here you get the code which you stashed.

* Git Role

# Administrator

# Integrator

# Individual developer(stand alone)

# Individual developer(participant)

* GIT Operations:
  + **Clone:** This is a startup process for working on repository where you are not the creator of the repo, you are a individual programmer contributing to repository.
    - “git clone source.git destFolderName”
    - “git clone <https://github.com/sachinsnale/LearnGit.git> LearnGit”
  + **Pull:** This command is used to get the latest changes made in particular branch, it may be master or task specific branch. Or it call be pulling the whole repository which consist of current branch changes + new branches added by other people.
    - “git pull master”
    - “git pull - -all”
    - “git pull origin branch\_name
  + **Branch:** This is where you start your work, you create new branch from exiting baselined master branch / or from any other branch and keep updating the code in branch as required by your task.
    - Only create branch : “git branch branchName”
    - Create and switch to branch: “git checkout –b branchname
  + **Add:** this command does the adding of new file to staging area and also the current changes which user wants to add for commit
    - Single file :“git add filename”
    - Multilple file “git add filename1 filename2
  + **Commit:** With this command you are updating your local repository as well as the local branch with some new files or may be some modification. It doesn’t update the server.
    - Single file-> “git commit filename”
    - Staged file-> “git commit”
    - All changes except untracked files-> “git commit -a”
  + **Push:** this is the command where you update the server with your changes, mostly it consist of updating your branch or all together pushing a new branch which was not available on the server.
    - “git push origin branch-name”
    - “pit push origin HEAD” (HEAD is nothing but current branch you are working on)
    - “git push” (the current branch commit will pushed if the branch is upstream/tracked)
  + **Checkout:** this command is used to switch in-between branches
    - “git checkout branch-name”
  + **Status:** This command is used to check which files are changed from in current branch, it will give list files which are modified.
    - “git status”
  + **Diff:** This command is used to get the difference of content of the changed files.
    - Diff with other branch: “git diff branchName”
    - Diff with current branch: “git diff”
  + **Fetch:** git fetch
  + **Stash:** This command is used to back up your local changes which will not be lost even if you move out from the current branch.
    - Stash the changes: “git stash”
    - Apply the changes:
      * “git apply stash”
      * “git pop stash”
      * “git drop stash”
  + **Merge:** This command is used to merge some other branch code into your branch and vice versa.
    - “git merge branch\_name”
* GIT tricks and tips:
  + Rename Branch :-
    - Step 1:
      * if you are on the branch you want to change
        + “git branch –m new-name”
      * if you are on different branch
        + “git branch –m oldname newname”
    - Step 2:
      * Delete the old-name remote branch and push the new-name local branch.
      * “git push origin :oldname newname”
    - Step 3:
      * Reset the upstream branch for the new-name local branch.
      * “git push origin -u newname”
* Setup git bash on windows

# Installing git

Ifm software Center : click on windows start Menu and type ”software Center”

Search “git” in software center and install git.

OR

Install the Git version from our share drive:

**I:\IT\Install\Git**

While installing it may ask for text editor, choose anyone that you are comfortable. Also add the git bash to start menu.

* Create your github account

Visit <https://github.com/> and sign up with your **personal mail id** as this will be your account on external git server.

Ifm wide we have our own git servers and you can use your official mail ID there.

* Git Bash

Git bash is a command line interface for git. All you need to use git is this command line interface and learn some commands.

# Proxy settings

To clone the repos from github.com we need to set up proxies so that we can access github.com from ifm network.

Open git bash and give following command

git config --global http.proxy http://proxyUsername:proxyPassword@proxy.server.com:port

git config --global https.proxy http://proxyUsername:proxyPassword@proxy.server.com:port

ifm proxies are available at

**I:\IT\HowTo\IT\_Fixes.txt**

# Setting username and email account:

**git** config --global user.name “your Name”

**git** config --global user.email “your external email” ( change this when you will be using ifm git accounts)

# Clone First repo

git clone <https://github.com/sachinsnale/LearnGit.git>