**Agenda**

**Slide\_one ( Git )**

**What**: open-source distributed version/source control utility to track code history.

**Why**: Improve developer operations by enhancing collaboration and code integration, saving hours of work, code delivery, and headaches.

**How** do we utilize it: CLI tools, and Github web service, Remote version control tools supporting git.

**Slide\_two ( Github )**

**What**: Remote git web service. Not only does it host your code, but allows substantial functionality including wiki, issue tracking, Agile Kanban project UI, etc.

**Why**: Utilization of massive amounts of functionalities available. Easy-to-use GUI for remote service. “FREE” FOOD.

**How** do we utilize: Booz Allen provides enterprise level [Github service](https://github.boozallencsn.com/) through [Smart Suite](https://ssp.boozallencsn.com/projects)

**Slide\_three ( Concept )**

**Working directory:** your local working directory

**Staging Area:** stage your files in this area. Status of staged files can be found using git status

**Git repo:** is the hidden repo available in your working directory

**Slide\_four ( Commands )**

|  |  |
| --- | --- |
| Command | Description |
| init | Initialize git repo |
| clone | Copy a remote git repo using https or ssh protocol |
| remote | Manage set of tracked repos |
| status | Check status of the git repository |
| log | Check log history of the git repo |
| add | Stage files to be committed to the git repo |
| commit | Commit your changes to the git repo |
| push | Push your code changes to the remote git repo |
| fetch | Fetch code from the remote git repo |
| merge | Merge remote repo code with your local working directory code, in your local working repo |
| pull | Fetch + Merge |
| reset | Uncommits your changes. --hard removes all staged files; --soft uncommits but retains your code in your working directory |
| revert | Replicates your current working directory code to mirror the commit you select to revert to while also maintaining your history |
| branch | List, create, or delete branches |
| checkout | Switch branches |

**Slide\_five ( Branch & Merge )**

**Branch**

* Ability to branch work to maintain the master branch
* Initial pointers: master, HEAD, other branch pointers
* Always create a branch during feature update and bug fix
* Never merge unfinished work. But once finished always merge to the stable branch and delete the old branch to keep branch repo clean and manageable
* Branches can also be created on release cycle basis

**Merge or rebase branches**

* Merge essentially merges code maintaining code history for both branches.
* Rebase is more helpful for complex merges by just adding you code on top of the branch you are merging with, merge will show you all conflicts at once. Rebase will show on a commit by commit basis. However the code history is affected <https://medium.com/datadriveninvestor/git-rebase-vs-merge-cc5199edd77c>

**Slide\_six ( Workflow )**

**Pull workflow**

* Git status
* Git fetch –all
* Git branch -a
* Git merge master blah
* Or a simple git pull is sufficient as well, if the code complexity is minimal and you have the utmost trust in your fellow developers and lead to have modularized the code

**Push workflow**

* Git status
* Git pull (resolve issue if needed, only proceed if safe)
* Git push

Use case for development and production environments: Set up a git code on dev and always pull on production this way you are ensuring that you are sitting actively on the server that may go offline due to file permissions or other code issues.

Other best practice:

* Always keep your working directories clean. Humans are a forgetful, computers can be ruthless.
* Never force push.
* Always pull the code before pushing or committing your changes. If you accidently commit and then pull changes and notice more changes have been made to the master, you can safely undo your changes to make sure you don’t cause any code conflicts. You can do this by soft reset -> stash -> merge -> fetch -> stash pop -> commit -> push.

**Slide\_seven (Helpful tips )**

**Git reset** unstages your files using git reset

Git reset done easy to completely reset your commit (short hashes are first 7 characters):

Usage -> git reset –hard commit-hash ; commands below help you reset commits back numerically

* 1 commit : git reset –hard HEAD^
* 2 commits: git reset –hard HEAD~2
* 3 commits: git reset –jard HEAD~3

Note: use this to show staging in action. Working directory unaffected due to reset.

**Git reverts** maintain code history, while allowing you to move back in time. Can be pretty messy in my opinion. Recommended branching before attempting dangerous time travel on master branch

**Alias**

1. git config –global alias.unstage ‘reset HEAD –’
2. git config –global alias.co checkout

**Config**

1. git config --global user.name "Your Name"
2. git config --global user.email you@example.com

**Git remote** allows you to add remote repos and keep track of it.

Git remote -v shows you all the remotes in your repo

Git remote show remote\_name shows you all information on the remote including branch differences

**Simple Exercise**

1. git init or clone
2. commit code
3. push code
4. pull changes from other devs

**Sequence of events**

1. repo created
2. changes made by dev are pushed with a file with their names one by one
3. changes by two devs are pushed at the same time, one should get rejected
4. same file is changed and push
5. same function is changed and push, the guy that pushes after hates his life