Version Control System

Introduction

This paper focuses on basic functionality of git, including a history, vocabulary, and commands. It is intended to show what concepts I learned using the training course *Git Essential Training* by Kevin Skoglund and should not be used as a comprehensive learning environment.

History

Git was developed by Linus Torvalds for the development of the LInux Kernel. He did not like the version control systems (VCSs) that were available to him, and he started the development of Git in 2005. Git will track changes to a document using a distributed version control layton, meaning that multiple people can work at the same file at the same time in a parallel fashion. As a result, there is no single master copy; by convention, most users designate a master copy, but it means that that network access is not necessary for development and it encourages forking on projects. Additionally, Git takes a snapshot of an entire folder's contents instead of tracking individually named files, and unlike other VCSs, git stores all of its tracking information in one folder at the top level. This makes it very easy to add and change the project configurations.

It is a superior version control system and it is completely free. It is open source, faster (up to 100x in some cases), and safe. Git uses checksums to ensure that the data matches up and maintains data integrity, and easily references commits using unique SHA values. Figure 1 shows basic commands for terminal git inputs.

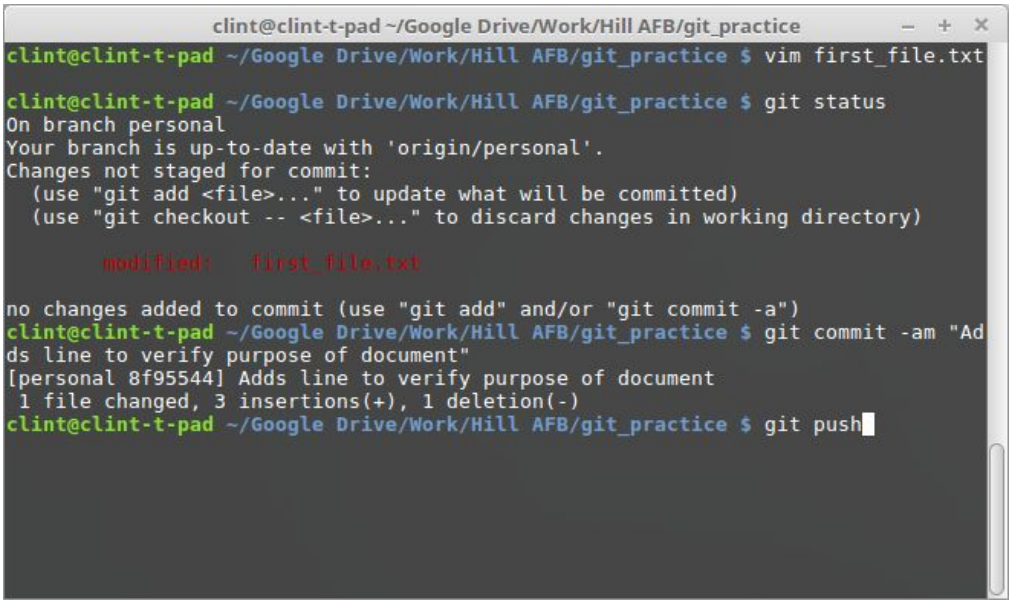

Figure 1

Figure 2 presents a view that helps visualize what is happening in git for the project workflow. To fulfill the requirement, I made a simple text file in a folder named git_practice, initialized a repository, and added it to a remote repository on github. As new features are required for a project, git tracks the branches and will display a log that shows the development of the project. The command seen in the box below is used to create the graphical representation.

```
$  git log --graph --oneline --all --decorate
```

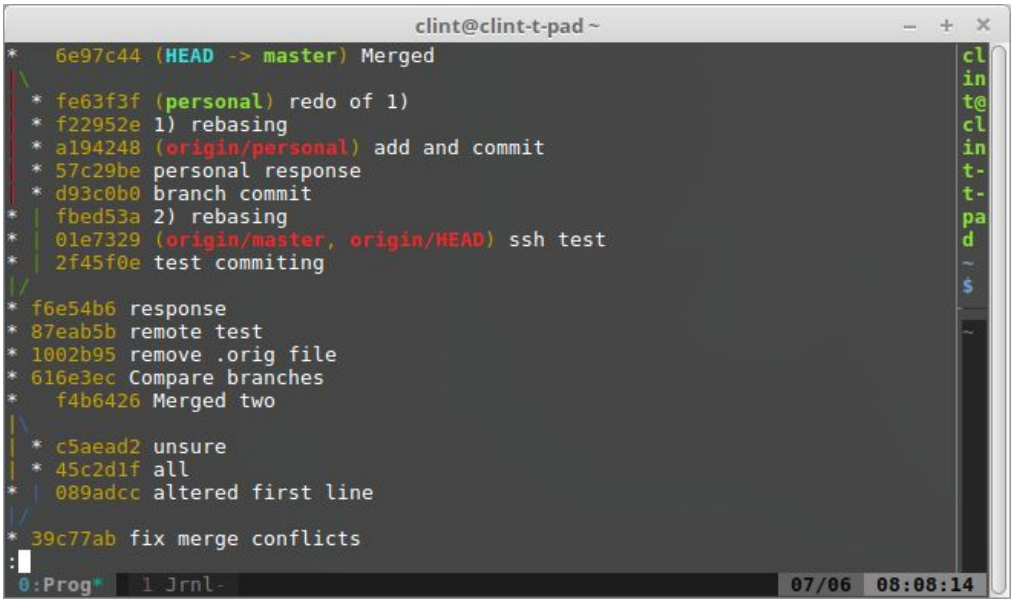

Figure 2: Graphical Representation.

Vocabulary Words:

Table 1: Vocabulary word used frequently in a git workflow

Word	Definition
Kernel:	The outermost part of the computer operating system handles all of the user and program I/O operations.
SHA:	A 40 digit number that refers to the current commit and is used for data validation
Head:	Where the current commit is pointed.
Merging:	Combining two branches. Vimdiff is my editor of choice.
Reverting:	Go back to a previous commit. Often results in merging.
Reset:	Moves the head pointer to wherever the user specifies.
Soft Reset:	Does not change staging index or working directory
Medium Reset:	Does not change working directory, but changes staging index to match reset point.
Hard reset:	Changes staging index and working directory.
Stash:	Temporarily saves files so that so that the user can move between branches
Rebasing:	Similar to merging, but results in a linear project history. It can come at the tradeoff of safety and traceability. It should never be used on public branches.
Remote:	refers to a remote repository where a user can push their repository changes. The branch origin/master tries to always keep in sync with the online repository
Fetch:	Make local copy of work synchronized with repository. Fetch before you push (to see if you can merge). Always fetch.
Configuration files:	Mergetool and .gitignore are examples of useful stored information, as well as alias set-ups.
Checking Out:	Moving between branches and reverting a single file.
Three Tree-architecture:	Has a staging index, and waits until user is ready to push up specific files.
Fast forward merge:	No reason to make a new commit, just make both branches point to the same SHA. (Point to same SHA)

Git commands

▸ General

git add .	Add all to staging directory
git commit -am " "	Add to staging and commit
git reset	unstage everything
git rm [name]	remove staged item
git dff	see the differences between saved and current
git log -n 4	Print commits to see change
git log --oneline	show only oneline
git status	State of the staging index
git clone <path> .	the "." tell git to copy into current empty folder.

▸ Undo

git checkout -- name.txt	revert current edits to most recent save (file only)
git checkout [branch]	change whole working directory into other.
git checkout SHA [file]	checkout old version of file

Reverting NOTE: Go back to an entire state of point

1) Run graphical representation (git logg), quit with :q
2) git revert SHA
3) merge

git clean -n	see what would be removed if not in staging index or tracked
git clean -f	remove all un-tracked files from repository.

▸ Branches

git branch -b name	make a new branch from the current
git checkout [name]	change to branch
git diff [name1] [name2]	compare two branches
git merge [name]	merge branch to current branch
git merge master feature	combines checkout and merge command
git branch -d [name]	delete branch
git merge --abort	abort the merge

STASHING

git stash save "message"	save current change to stash
git stash list	
git stash show -p stash@{0}	
git stash pop	No longer in stash in working directory
git stash drop stash@{0}	Delete stash

▸ vimdiff

git config diff.tool vimdiff	allow you to see differences with vimdiff

(after git fetch)

git (diff | difftool) master origin/master

git config --global merge.tool vimdiff
git mergetool
[c next difference
NOTE: you can type di<TAB> then RE... etc.
:diffg RE " get from REMOTE
:diffg BA " get from BASE
:diffg LO " get from LOCAL
:xa close all

▸ Configuration
git config merge.tool vimdiff set vimdiff as default mergetool
git config merge.conflictstyle diff3 see common ancestor as well.
vim .gitignore put in home folder, and write each type on each line

▸ Remotes
git remote list all current remotes
git remote add [name] <url> the remote origin is used by convention
git remote rm remove remote
git push -u origin master push current code to remote. -u
git fetch syncs online remote with local
git merge origin/master does what git pull does

git push <url>
git fetch <url>

git branch -r see remote branches
git checkout -b [branch] [origin/branch] checkout branch from github

▸ Bare repositories
mkdir name.git; cd name.git
git config receive.denyCurrentBranch ignore
git init --bare --shared=group

▸ Graphical representation
git log --graph --oneline --all --decorate

Conclusion

I encountered two main problems as I was learning git. 1) I did not realize that the command *get checkout* is used to both change to a different branch, as well as revert a current working filer to a previous version. 2) To avoid merge conflicts it is best to make short lines of code, keep

commits small and focused, and avoid edits to white space and other other similar unintentional edits. It is important to merge as often as possible so that merging versions is not a significant task in the end.