A Quick Guide to Git

1 Before Version Control System

1.1 diff

This is the below command used to check the difference between two files

```
$ diff file1 file2
$ diff -u file1 file2
$ diff -u file1 file2 > change.diff
```

Here `cheange.diff` can also be called as *patch file*. There are other GUI based editors available

- meld
- vimdiff
- KDiff3

1.2 patch

This command patches the difference between the file will be patched

```
$ patch cpu_usage.py < cpu_usage.dff
```

The above command will patch the difference found in the `cpu_usage.dff` into `cpu_usage.py`

2 Version Control System

VCS comes in Centralized and Distributed architecture. `Git` is distributed VCS. `GitHub`, `GitLab`, `BitBucket` etc... can be used to host codes online to collaborate better. `Git-SCM` stands for git **Source Control Management** = **VCS**.

2.1 Download

you can downlaod the git from the following URL:*https* : *//gitforwindows.org/*

3 Git Configuration

```
$ git config --global user.name "bjnaga@gmail.com"
$ git config --global user.email "Naga Ganesh B J"
# Enables helpful colorization of command line output.
$ git config --global color.ui auto
# show the current configuration
$ git config -l
```

4 Creating Git repositories

Initialize the git repository. Default branch is Master or main or anyother name given when user has given while installing the software.

```
$ git init
```

4.1 Tracking a file

Untracked files to staged file to Tracked file. `git status` can be used to check the status of the file for staged(also called index) or unstaged.

```
# adding . will add all the files into staging
# unless mentioned in gitignore file.
$ git add .
$ git add filename
```

4.2 Commit

Git directory : History of all the files and changes to it.
Working Tree : Current state of the project including any changes that we have made.
Staging Area : Contents that have been marked to be included in to next commit.
Trackfiles are part of snapshot, Untracked files aren't part of the snapshot. Each tracked file can be in one of the 3 states modified, staged , committed. Staging area files are added, modifies and untracked files are not committed.

```
# the below command will open a file in default
# editor selected during installation to enter
# a commit message.
$ git commit
$ git commit -m "short-one-line-commit-message"
```

once we commit git takes a snapshot by annotating the code state.

4.3 Anatomy of commit messages

Follow commit message standards to commit codes future readability. Normally commit can be broken up into a few sections "short summary of commit, followed by a black line, Full description of the changes details why it is necessary or anything interesting about them or difficult to understand". **eg:**

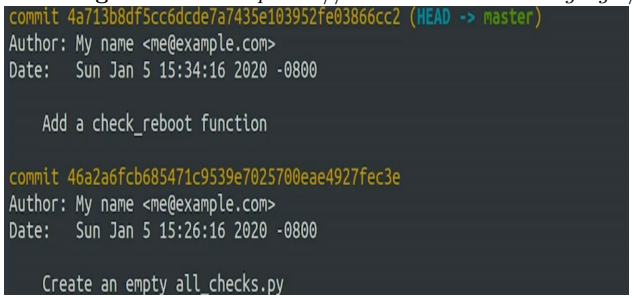
```
Provide a good commit message example

The purpose of this commit is to provide an example
of a hand-crafted, artisanal commit message. The
first line is a short, approximately 50 character
summary, followed by an empty line. The subsequent
paragraphs are jam-packed with descriptive
information about the change, but each line is kept
under 72 characters in length.

If even more information is needed to explain the
change, more paragraphs can be added after blank
lines, with links to issues, tickets, or bugs.
Remember that in future you will thank your current
you for your thoughtfulness and foresight!
# This is a comment and will be ignored, short commit
# messaged will be difficult to have context
```

Git commit style resources including Linux kernel kernel documentation and other resources.

4.4 history of commit

```
$ git log
```

will give us 40 digit SHA hash to uniquely identify each commit, author and email of each commit, Date and time of commit and commit message. **Link for Practice lab on Google Cloud** *https* : *//www.cloudskillsboost.google/*



5 Using git locally

5.1 advanced git interaction

5.1.1 Skipping to staging area

```
$ git commit -a
```

Above command is a shortcut to stage any changes to tracked files and commit them in one step. Un-tracked files are not committed(new files will not be committed). We will have to track the untracked file first by using `$ git add file-name.txt` after this command file-name.txt will be tracked by git and can be committed automatically by skipping the add option to git.

5.1.2 Git HEAD

Git uses the HEAD alias to represent the currently checked-out snapshot of your project. It is more useful to know where we are currently at(point the current branch). Its more useful during rollback(commit undo).

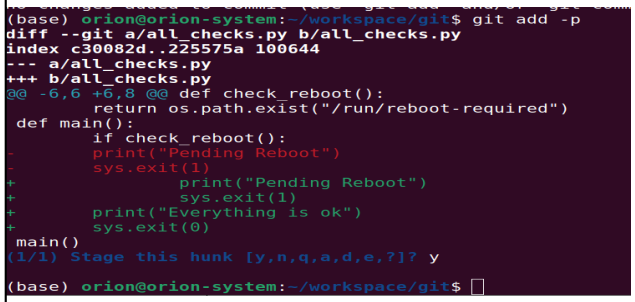
5.1.3 Getting more information about the changes

`diff -u` is equivalent to`git log -p` this will tell you what was added and removed in previous commit(this is patch information of each commit). Git automatically displays this information using paging tool to view the contents as it will generally long. It can be **exited by pressing q**.

```
$ git log -p
# git show commit-id
$ git show 21ab7787d72ab13c6c3ef8b8668c7fe3dd69abce
# will show the perticular commit information

$ git log --stats
# Shows more information about what files were edited
# how many lines were added and removed
# more options are available
$ git help -a
$ git help -g
```

The last two commands provides gist of git available subcommands and git concepts guides.

```
$ git diff
# the above command is equal to diff -u
# it is used to record the changes to the file
# the difference between commited file to unstages
# file will be shown
# diff - Show changes between commits, commit and
#         working tree, etc

$ git add -p
# Above commands show the changes and prompts if
# would like to commit those changes in case
# we would like to check before staging
```



`git diff` only show difference between unstages and committed by default.

`git diff --staged` for difference between staged but not committed.

5.1.4 Deleting and Renaming a file

Deleting We can remove the file from the git using `git rm` this command will remove the file from tracking(Remove files from the working tree and from the index) once the file is removes the removed file will automatically be in staging area so we will have to commit the code with a message to save the commit.

Renaming or moving a file `git mv` We can use this command for renaming and moving files between directories.

Ignoring a file This is used to ignore all the log files and other OS generated files(temp files) that we would like to ignore. `.gitignore` will contains the rules for files to be ignored while committing for current repo. This .gitignore files has to be tracked and commited in the repository.

5.2 undoing things

Undoing unstaged changes

`git checkout file-name.txt` this will all the changes done to the file-name.txt and the version committed will be used.

`git checkout -p filename.txt` this will show all the changes to the file and you will be able to do appropriate things if there are more than a few changes.

Undoing staged changes

`git restore --staged file-name.txt` This will unstage the file-name.txt to unstaged state. This command is opposite to `git add`. we add -p options to select which specific changes do we need to reset.

5.3 rollback

Used to rollback to previous committed snapshot. Earlier we used to use `git revert` now we can use `git restore` instead

```
$ git revert HEAD
# The above command can only revert
# back to previous commit, if we
# want to revert back to older commit
# we will have to follow other aproach
$ git log -p -2
# will show the log history of last 2
# commit
```

git reset vs revert vs rebase vs restore

git restore "restore is about restoring files in the working tree from either the index or another commit. This command does not update your branch. The command can also be used to restore files in the index from another commit."

git revert" revert is about making a new commit that reverts the changes made by other commits."

5.4 identifying a commit

every git commit is identified by a commit id 90930173e9837386235e82af8668238a2158604a which is a SHA1(cryptographic hash algorithm) 40 character. It is base on commit message date snapshot author of the working tree. The hash of two collision producing same hash is called *collision* and is very very slim. You can't change anything without changing the SHA1 hash for the commit. `git commit --amend` The git commit –amend command is a convenient way to modify the most recent commit. It lets you combine staged changes with the previous commit instead of creating an entirely new commit. It can also be used to simply edit the previous commit message without changing its snapshot. But, amending does not just alter the most recent commit, it replaces it entirely, meaning the amended commit will be a new entity with its own ref. To Git, it will look like a brand new commit, which is visualized with an asterisk (*) in the diagram below.

Don't amend public commits

Amended commits are actually entirely new commits and the previous commit will no longer be on your current branch. This has the same consequences as resetting a public snapshot. Avoid amending a commit that other developers have based their work on. This

is a confusing situation for developers to be in and it's complicated to recover from.*https* : *//www.atlassian.com/git/tutorials/rewriting-history*

6 Branching and Merging