A MINI TUTORIAL

Aditya Gore

September 7, 2016

Configure the User Settings

1. The commands that are used to configure
   - System wide configuration
     ```
     git config --system
     ```
   - User based configuration
     ```
     git config --global
     ```
   - Project based configuration
     ```
     git config
     ```
2. The variables that we can configure are
   - User name
     ```
     git config --global user.name "Your name"
     ```
   - User email address
     ```
     git config --global user.emal "Your email address"
     ```
   - Custom editor
     ```
     git config --global core.editor "Editor"
     ```
   - Git bash colored user interface
     ```
     git config --global color.ui true
     ```
3. To view the configurations
   ```
   git config --list
   ```

Create Repository w/o Github

1. If you want to start off with the project without creating a repository on Github then create the directory called “xyz” where you want to start the project and then use git bash to navigate to the folder and once in there use the command

   ```
   git init
   ```

   This command will start tracking any changes that are made in the “xyz” folder from here on.

Create Repository on Github

1. Go to `https://github.com/` and create a repository “xyz”.
2. Once created click on clone or download button and get a url link to the project which will look like `https://github.com/yourname/xyz.git`.
3. Open the github bash in folder where you want the project and use the following command to clone the project.

   ```
   git clone https://github.com/yourname/xyz.git
   ```

4. This will create a folder called “xyz” in the directory you used the command.

Making commits

1. To add all the changes to the “staged” index use the command

   ```
   git add .
   ```

2. To commit the changes to the same “branch”

   ```
   git commit –m "Write down the message about the commit"
   ```

3. To remove a deleted file (working directory) from the repository

   ```
   git rm file_name
   ```

4. To move or rename a file

   ```
   git mv firstname secondname
   ```

5. To add and commit at the same time

   ```
   git commit –am "write the message here"
   ```

Viewing commits and logs

1. To view commits made so far

```
git log
```

2. To view past 3 commits

```
git log -n 3
```

3. We can use time filter to view the commits

```
git log --since=2016-09-15 --until==2016-09-22
```

4. We can use author filter to view the commits

```
git log --author="Aditya"
```

   The author can be matched on partial string but is case sensitive

5. To filter by words in the commit message

```
git log --grep="regexp"
```

6. To check the commits from “HEAD” going backwards.

```
git log HEAD
```

7. Useful options to use with git log

```
git log --oneline
git log --oneline -3 (previous 3 commits)
git log --oneline HEAD~3..HEAD~1
git log HEAD~10.. filename (to see all the commits made for that file)
git log -p HEAD~10.. filename (to see the changes to the file)
git log --stat --summary
git log --format=oneline
git log --format=short
git log --format=full
git log --format=fuller
git log --format=email
git log --graph
git log --oneline --graph --all --decorate
```

8. To see the changes from a previous commit

```
git show HEAD~2
```

9. To compare the commits

```
git diff HEAD~2 filename
git diff HEAD~3..HEAD~1 filename
```

Managing repositories/commits

1. To check the status of the current branch

```
git status
```

It gives the difference between the working directory, staging index and git repository.

2. To see the differences between the repository and working directory, file by file

```
git diff filename<optional>
```

3. To view the differences between the git repository and staging index, file by file

```
git diff --staged
```

4. To view changes side by side

```
git diff --color-words filename
```

5. Undo changes in the working directory

```
git checkout -- filename
```

This will checkout the the files from the repository at the current branch to the working directory.

6. Undo changes in the staging index

```
git reset HEAD filename
```

This will remove the file from staging index but would not modify the file in the working directory. If filename is not provided then it will remove all the files from the staging index without modifying the files in the working directory.

7. To amend the commit (can only change the most recent commit)

```
git add filename
git commit --amend -m "write the message from the recent commit"
```

This changes the most recent commit (the one that HEAD points to)

8. To revert changes made to a file to an older commit

```
git checkout 2afd5fd3b0 -- filename
git commit -m "Revert the changes to an older commit"
```

2afd5fd3b0 is the SHA for the commit where you want to revert to. The first command will checkout the file in the staging index.

9. To revert to the changes in one single step

```
git revert 2afd5fd3b0
```

10. Another way to undo a commit.

```
git reset --soft 2afd5fd3b0
```

The --soft option just moves the HEAD pointer to the repository. The staging index and the working directory remains unchanged.
The --mixed option moves the HEAD pointer to the repository and also modifies the staging index to reflect the repository. It does not change the working directory.
The --hard option moves the HEAD pointer to the repository and makes the changes to the staging index and working directory to reflect the repository.

11. Removing untracked files

```
git clean -n
git clean -f
```

-n tells you what it would do. -f forces the command to remove the untracked files.

12. To ignore tracked files

```
git rm --cached filename
```

First remove it from the staging index. Then add the filename to the .gitignore file.

13. To track an empty directory Add a file called .gitkeep

Referencing commits

1. To see the the tree-ish structure of commit

```
git ls-tree HEAD~2
```

This will show you the list of files in grand parent (second parent) of the commit where HEAD is pointing at.