Git - 2 (11 Mar 22)

Friday, March 11, 2022 8:59 PM

Pre-Class

In-Class

Git Cheat Sheet

Setup

Set the name and email that will be attached to your commits and tags

```
$ git config --global user.name "Danny Adams"
$ git config --global user.email "my-email@gmail.com"
```

Start a Project

Create a local repo (omit <directory> to initialise the current directory as a git repo

```
$ git init <directory>
```

Download a remote repo

```
$ git clone <url>
```

Make a Change

Add a file to staging

```
$ git add <file>
```

Stage all files

```
$ git add .
```

Commit all staged files to git

```
$ git commit -m "commit message"
```

Add all changes made to tracked files & commit

```
$ git commit -am "commit message"
```

Basic Concepts

main: default development branch
origin: default upstream repo
HEAD: current branch
HEAD^: parent of HEAD
HEAD~4: great-great grandparent of HEAD

By @DoableDanny

Branches

List all local branches. Add -r flag to show all remote branches. -a flag for all branches.

```
$ git branch
```

Create a new branch

```
$ git branch <new-branch>
```

Switch to a branch & update the working directory

```
$ git checkout <branch>
```

Create a new branch and switch to it

```
$ git checkout -b <new-branch>
```

Delete a merged branch

```
$ git branch -d <branch>
```

Delete a branch, whether merged or not

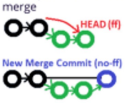
```
$ git branch -D <branch>
```

Add a tag to current commit (often used for new version releases)

```
$ git tag <tag-name>
```

Merging

Merge branch a into branch b. Add --no-ff option for no-fast-forward merge



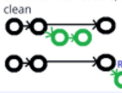

```
$ git checkout b
$ git merge a
```

Merge & squash all commits into one new commit

```
$ git merge --squash a
```

Rebasing

Rebase feature branch onto main (to incorporate new changes made to main). Prevents unnecessary merge commits into feature, keeping history clean




```
$ git checkout feature
$ git rebase main
```

Interatively clean up a branches commits before rebasing onto main

```
$ git rebase -i main
```

Interatively rebase the last 3 commits on current branch

```
$ git rebase -i Head~3
```

Undoing Things

Move (&/or rename) a file & stage move

```
$ git mv <existing_path> <new_path>
```

Remove a file from working directory & staging area, then stage the removal

```
$ git rm <file>
```

Remove from staging area only

```
$ git rm --cached <file>
```

View a previous commit (READ only)

```
$ git checkout <commit_ID>
```

Create a new commit, reverting the changes from a specified commit

```
$ git revert <commit_ID>
```

Go back to a previous commit & delete all commits ahead of it (revert is safer). Add --hard flag to also delete workspace changes (BE VERY CAREFUL)

```
$ git reset <commit_ID>
```

Review your Repo

List new or modified files not yet committed

```
$ git status
```

List commit history, with respective IDs

```
$ git log --oneline
```

Show changes to unstaged files. For changes to staged files, add --cached option

```
$ git diff
```

Show changes between two commits

```
$ git diff commit1_ID commit2_ID
```

Stashing

Store modified & staged changes. To include untracked files, add -u flag. For untracked & ignored files, add -a flag.

```
$ git stash
```

As above, but add a comment.

```
$ git stash save "comment"
```

Partial stash. Stash just a single file, a collection of files, or individual changes from within files

```
$ git stash -p
```

List all stashes

```
$ git stash list
```

Re-apply the stash without deleting it

```
$ git stash apply
```

Re-apply the stash at index 2, then delete it from the stash list. Omit stash@{n} to pop the most recent stash.

```
$ git stash pop stash@{2}
```

Show the diff summary of stash 1. Pass the -p flag to see the full diff.

```
$ git stash show stash@{1}
```

Delete stash at index 1. Omit stash@{n} to delete last stash made

```
$ git stash drop stash@{1}
```

Delete all stashes

```
$ git stash clear
```

Synchronizing

Add a remote repo

```
$ git remote add <alias> <url>
```

View all remote connections. Add -v flag to view urls.

```
$ git remote
```

Remove a connection

```
$ git remote remove <alias>
```

Rename a connection

```
$ git remote rename <old> <new>
```

Fetch all branches from remote repo (no merge)

```
$ git fetch <alias>
```

Fetch a specific branch

```
$ git fetch <alias> <branch>
```

Fetch the remote repo's copy of the current branch, then merge

```
$ git pull
```

Move (rebase) your local changes onto the top of new changes made to the remote repo (for clean, linear history)

```
$ git pull --rebase <alias>
```

Upload local content to remote repo

```
$ git push <alias>
```

Upload to a branch (can then pull request)

```
$ git push <alias> <branch>
```

Recap

Recap-What is Git?

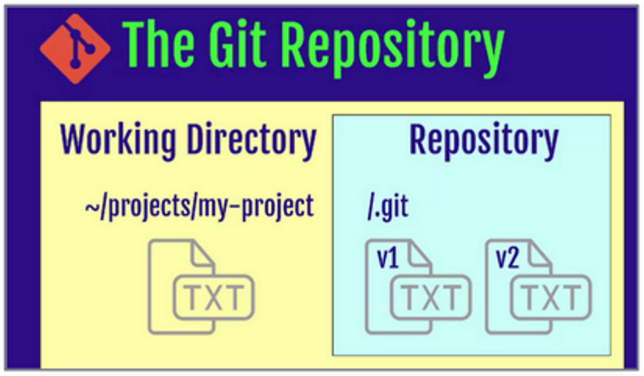
- **Git** is an **open source distributed version control system**
- **Tracks** and **records** changes to files over time (**versioning**)
- Can **retrieve** previous version of files at any time (**time travel**)
- Can be used **locally**, or **collaboratively** with others (**teamwork**)
- Contains extra information such as **date**, **author**, and **a message explaining the change**
- **Compare** and **Blame**
  - What changed
  - When it changed
  - Why it changed
  - Who changed it

Recap-Git Repository

What is a repository

- A directory or storage space where your projects can live.

- Local Repository
- Remote Repository (Central Repository)



Recap-Workflow-Git's "three trees"

Working Directory

Where you work.Create new files, edit files delete files etc.

Staging Area (Index)

Before taking a snapshot, you're taking the files to a stage. Ready files to be committed.

Repository (Commit Tree)

Committed snapshots of your project will be stored here with a full version history.

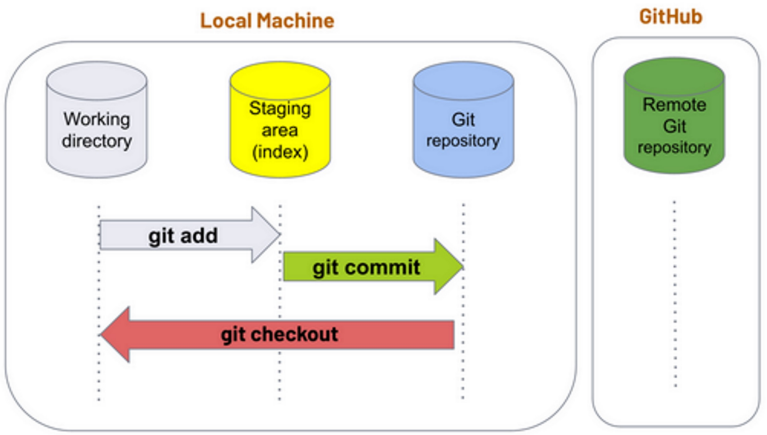
Recap-Git Config

- Git needs your identity to mark/label changes / editor

```
git config --global user.name "Your Name"
git config --global user.email "Your Name"
git config --global core.editor "vim"
git config --list
```

Recap-Basic Commands

git help
git init
git status
git add .
git rm --cached
git commit -m "abc"
git log
git checkout commitID



Recap-Tasks

Task-1

- Create a new repo under **my-second-project** folder
- Create a file named **file1.txt**
- Change the file
- Stage the file
- Commit the file to your repo

Task-2

- Create a file named **file2.txt**
- Edit **file2.txt**
- Stage
- Delete the file **file1.txt**
- Rename **file2.txt** >> **file3.txt**
- Stage **file3.txt**
- Unstage **file3.txt**
- Stage **file3.txt** again
- Commit the file to your repo
- Change the message of the commit
- Switch back to your first commit in Task-1

Recap-Solutions

Create a new repo under my-second-project folder	`git init`
Create a file named file1.txt	`touch file1.txt`
Change the file	`vim file1.txt`
Stage the file	`git add .`
Commit the file to your repo	`git commit -m "message"`
Create a file named file2.txt	`touch file2.txt`
Edit file2.txt	`vim file2.txt`
Stage	`git add .`
Delete the file file1.txt	`rm file1.txt`
Rename file2.txt >> file3.txt	`mv file2.txt file3.txt`

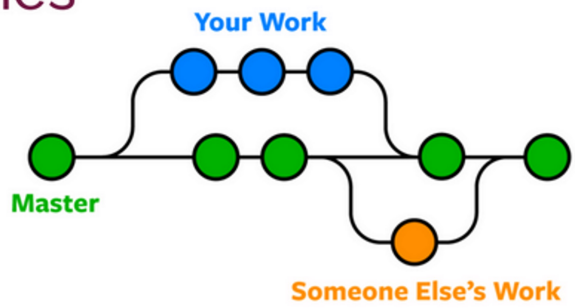
Git Branches

"Github changed it default branch name from master to main after the Black Lives Matter incident."

Snapshot of current commit

Master branch: original copy of book
Branch: pirate copy

Branches



- Production of the project lives on master/main branch
- Branches are reference to a commit

```
Erics-Mac:project eric$ git branch
* master
```

naming convention is set before a project starts

Branches

- to see local branches

```
git branch
```

- to see remote branches

```
git branch -r
```

- to see all branches

```
git branch -a
```

Creating/switching branches

- create a new branch

```
git branch Branch name
```

- switch to a branch

```
git checkout Branch name
```

- create a new branch and switch to that branch

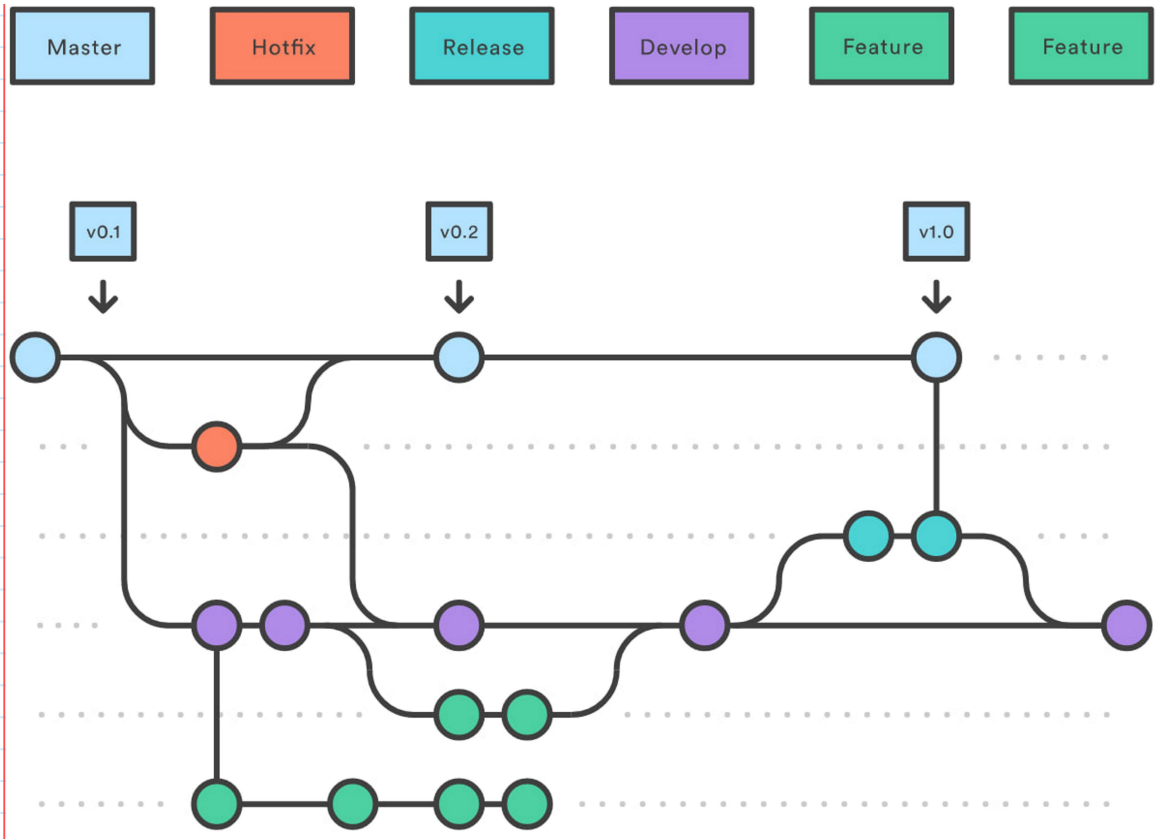
```
git checkout -b Branch name
```

expected behavior. Git will not create a `master` branch until you commit something.

git checkout -b test2 = git branch test2 && git checkout test2

- delete a local branch

```
git branch -d Branch name
git branch -D Branch name
```



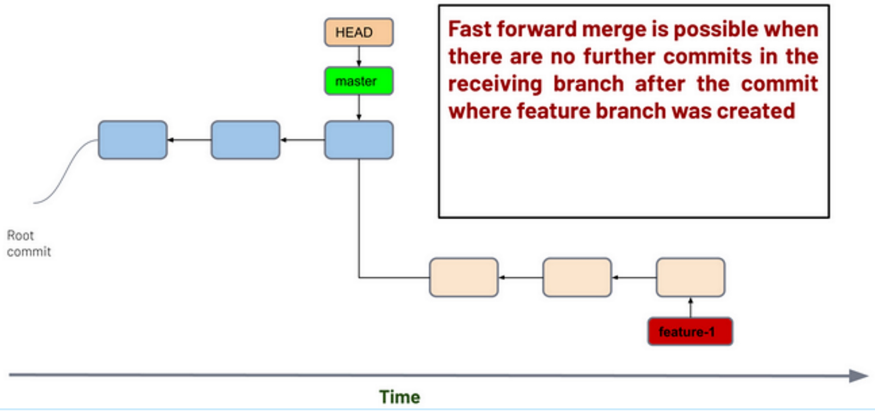
git merge

- fast-forward
- three-way merge

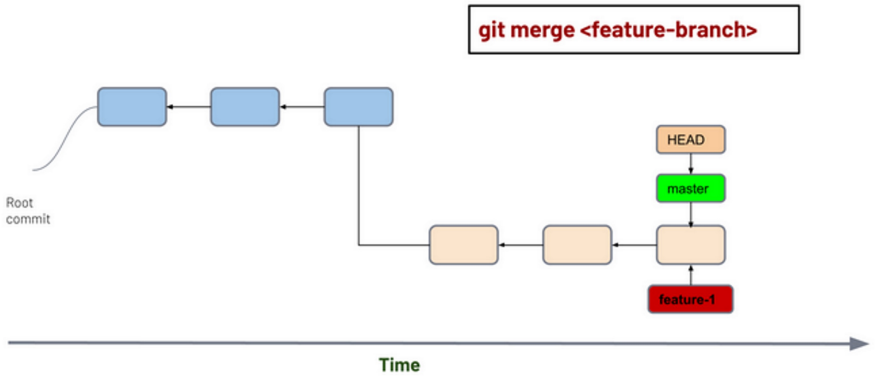
git log --all --graph

```
ubuntu@marcus: ~/project/handson_project
*   commit 5e6b1991c6dc2c04ca3c5e3a17c2cf82bd8d9bfd (HEAD -> develop)
|\  Merge: 74cd65a ac40d22
| | Author: marcus <marcus@clarusway.com>
| | Date:   Fri Mar 11 21:45:04 2022 +0100
| |
| |     index.html was updated and conflict was resolved
| |
| * commit ac40d22eb03f1f27b4af518ec326cbe828454880 (feauture)
| | Author: marcus <marcus@clarusway.com>
| | Date:   Fri Mar 11 21:33:45 2022 +0100
| |
| |     index.html updated in feature branch
| |
| * commit fc34bd2846f3d45a08400b305253a8c39643a6ef
| | Author: marcus <marcus@clarusway.com>
| | Date:   Fri Mar 11 21:31:51 2022 +0100
| |
| |     feat:index.js was added
| |
* | commit 74cd65aaf04fbe4c697d03eaeb9ca735be496314
|/  Author: marcus <marcus@clarusway.com>
|   Date:   Fri Mar 11 21:25:31 2022 +0100
|
|       index.html was updated
|
* commit 81190767fb6185b8cd2f8d8fdba2f0dae6c56cfb
| Author: marcus <marcus@clarusway.com>
| Date:   Fri Mar 11 21:23:31 2022 +0100
|
|     Dockerfile was added
|
* commit 69447e5c2afa41acaa04e5f897da6bff41945d08 (master, hotfix)
| Author: marcus <marcus@clarusway.com>
| Date:   Fri Mar 11 21:14:43 2022 +0100
|
```

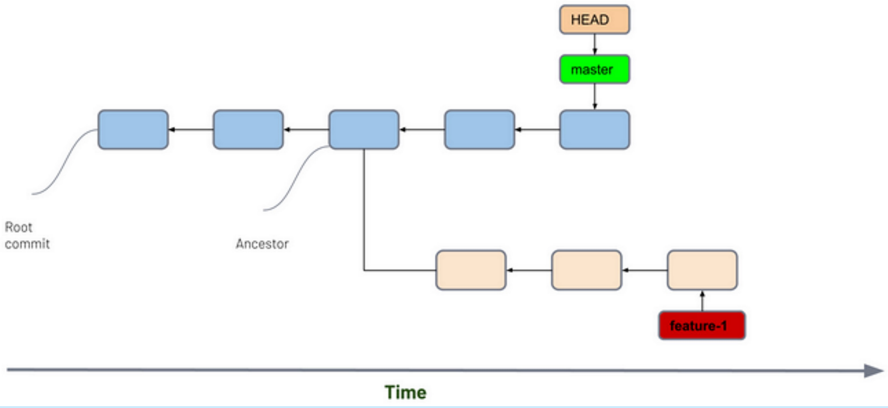
Fast forward merge



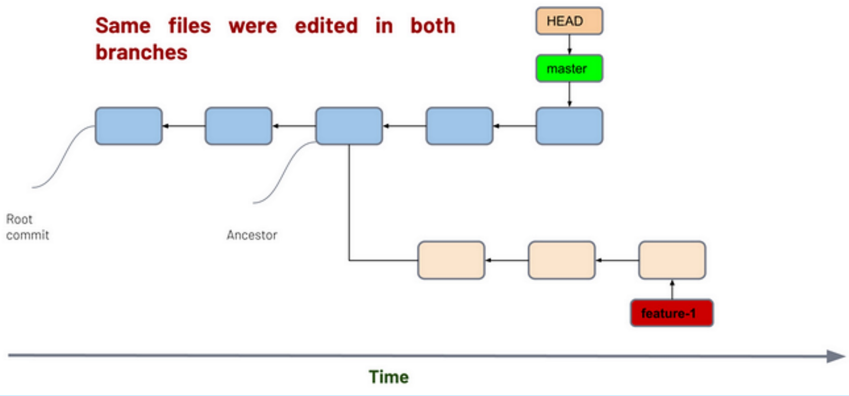
Fast forward merge



3-way merge

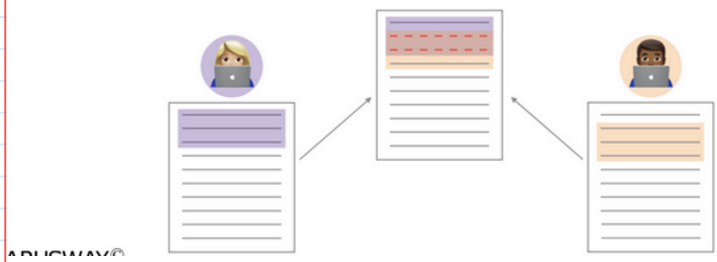


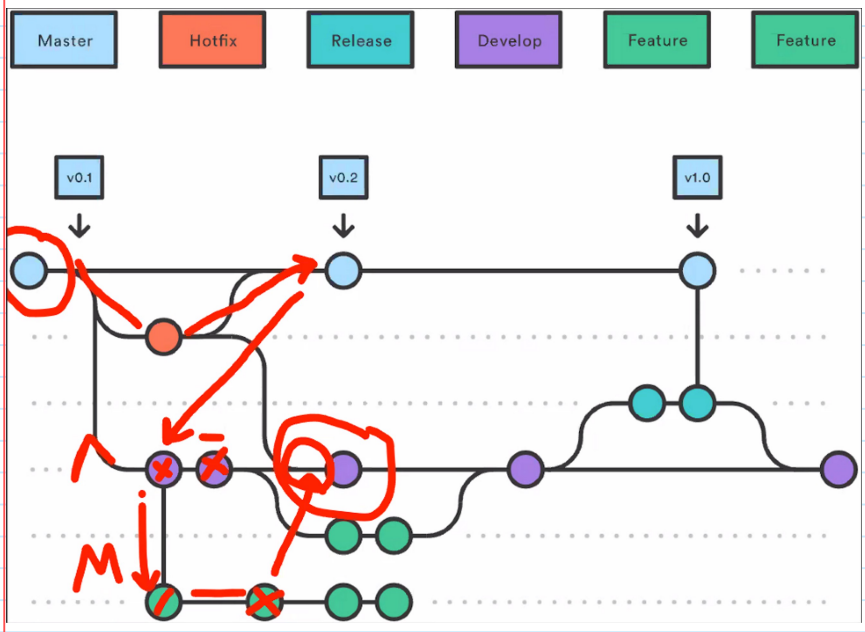
Merge Conflicts



Github - Merge Conflict

→ **Merge conflicts** happen when you merge branches that have competing commits, and Git needs your help to decide which changes to incorporate in the final merge.





History

```
1387  pwd
1388  git config --global user.name "edelweiss"
1389  git config --global user.email "edelweiss@edelweiss.com"
1390  git config --global core.editor "nano"
1391  git config --list
1392  git help
1393  git init help
1394  git init --help
1395  git status --help
1396  ll
1397  rm -rvf help
1398  ll
1399  rm -rfv project
1400  ll
1401  mkdir project
1402  cd project/
1403  mkdir first_project
1404  cd first_project/
```

```
1405  git init
1406  git status
1407  touch index.html
1408  git status
1409  git add .
1410  git status
1411  git rm --cached index.html
1412  git status
1413  git commit -m "feat:index.html file was added"
1414  git add .
1415  git commit -m "feat:index.html file was added"
1416  git status
1417  git log
1418  ll
1419  nano index.html
1420  git status
1421  git add .
1422  git commit -m "feat:index.html file was updated"
1423  git log
1424  clear
1425  ll
1426  cat index.html
1427  git log
1428  git checkout 0993b
1429  ll
1430  cat index.html
1431  git log -a
1432  git log --help
1433  git log --all
1434  git checkout 5dba5
1435  cat index.html
1436  git log --all
1437  cd ..
1438  mkdir my-second-project
1439  cd my-second-project/
1440  git init
1441  ll
1442  touch file1.txt
1443  nano file1.txt
1444  git status
1445  git add .
1446  git commit -m "feat:file1.txt was added to repo"
1447  git commit
1448  git log
1449  git status
1450  history
1451  clear
1452  nano file2.txt
1453  git add .
1454  rm file1.txt
1455  mv file2.txt file3.txt
1456  git add .
1457  git rm --cached file3.txt
1458  git add .
1459  git commit -m "feat:file3.txt was added"
1460  git commit --amend
1461  git log
1462  git checkout 9c47e
1463  git status
1464  git log
```

```
1465  history
1466  clear
1467  cd ..
1468  git branch --help
1469  mkdir hands-on_project
1470  cd hands-on_project/
1471  git init
1472  touch index.html
1473  nano index.html
1474  git add .
1475  git commit -m "project initiated"
1476  git status
1477  git branch hotfix
1478  git checkout hotfix
1479  git status
1480  git branch
1481  ll
1482  nano index.html
1483  git status
1484  git add index.html
1485  git commit -m "fix: title was updated as Clarusway"
1486  git status
1487  clear
1488  git branch
1489  git checkout master
1490  git merge hotfix
1491  git log --all
1492  git checkout -b develop
1493  ll
1494  nano Dockerfile
1495  git status
1496  git add Dockerfile
1497  git commit -m "Dockerfile was added"
1498  nano index.html
1499  fg
1500  git add index.html
1501  git commit -m "index file inserted second para"
1502  git log --graph
1503  git checkout 9e521
1504  git log
1505  git checkout -b feature
1506  git log --graph
1507  nano index.js
1508  ll
1509  git add index.js
1510  git commit -m "feat:index.js added"
1511  nano index.html
1512  git add .
1513  git commit -m "index.html updated in feature branch"
1514  git branch
1515  git checkout develop
1516  git log --all
1517  git log --all --graph
1518  exit
```

Post-Class

- Additional resource:

Git beginner’s Cheatsheet

https://medium.com/@shanikae/git-beginners-cheatsheet-6abbc15f108b

★ Watch

Nano Editörü Nasıl Kullanılır?

git-cheat-sheet…

GitHub
GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

SETUP

Configuring user information used across all local repositories

`git config --global user.name "[firstname lastname]"`
set a name that is identifiable for credit when review version history
`git config --global user.email "[valid-email]"`
set an email address that will be associated with each history marker
`git config --global color.ui auto`
set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

`git init`
initialize an existing directory as a Git repository
`git clone [url]`
retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

`git status`
show modified files in working directory, staged for your next commit
`git add [file]`
add a file as it looks now to your next commit (stage)
`git reset [file]`
unstage a file while retaining the changes in working directory
`git diff`
diff of what is changed but not staged
`git diff --staged`
diff of what is staged but not yet committed
`git commit -m "[descriptive message]"`
commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

`git branch`
list your branches. a * will appear next to the currently active branch
`git branch [branch-name]`
create a new branch at the current commit
`git checkout`
switch to another branch and check it out into your working directory
`git merge [branch]`
merge the specified branch's history into the current one
`git log`
show all commits in the current branch's history

INSPECT & COMPARE

Examining logs, diffs and object information

`git log`
show the commit history for the currently active branch
`git log branchB..branchA`
show the commits on branchA that are not on branchB
`git log --follow [file]`
show the commits that changed file, even across renames
`git diff branchB...branchA`
show the diff of what is in branchA that is not in branchB
`git show [SHA]`
show any object in Git in human-readable format

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

`git remote add [alias] [url]`
add a git URL as an alias
`git fetch [alias]`
fetch down all the branches from that Git remote
`git merge [alias]/[branch]`
merge a remote branch into your current branch to bring it up to date
`git push [alias] [branch]`
Transmit local branch commits to the remote repository branch
`git pull`
fetch and merge any commits from the tracking remote branch

TRACKING PATH CHANGES

Versioning file removes and path changes

`git rm [file]`
delete the file from project and stage the removal for commit
`git mv [existing-path] [new-path]`
change an existing file path and stage the move
`git log --stat -M`
show all commit logs with indication of any paths that moved

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

`git rebase [branch]`
apply any commits of current branch ahead of specified one
`git reset --hard [commit]`
clear staging area, rewrite working tree from specified commit

IGNORING PATTERNS

Preventing unintentional staging or commiting of files

`logs/` **`*.notes`** **`pattern*/`**
Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.
`git config --global core.excludesfile [file]`
system wide ignore pattern for all local repositories

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

`git stash`
Save modified and staged changes
`git stash list`
list stack-order of stashed file changes
`git stash pop`
write working from top of stash stack
`git stash drop`
discard the changes from top of stash stack