# git workshop - dev boost

git book

## basic git exercise

Windows: use Git Bash

Linux/Mac: use the Terminal (possibly with Zsh+oh my zsh)

Commands we used:

- Git init
- Git config -l
- Git config -l –show-scope –show-origin
- Git config –global user.name david
- Git status
- Git add
- Git restore
- Git restore --staged
- Git commit
- Git commit -m
- Git commit -a
- Git commit -am
- Git log
- Git log --oneline
- Git log -p
- Git log --oneline -p
- Git log -3
- Git diff
- Git diff --staged

Self study:

- linux
  - command for changing a file name: mv
- git:
  - git rm ( = rm + git add)
  - git mv

Terms:

- working directory (AKA working copy OR working tree)
- staging area (AKA index)
- .git repository

Exercise:

- create a git repo (mkdir + cd + git init)
- add a file with some text (write to file then git add)
- Commit (git commit)
- modify file and quickly commit with git commit -am
- add another file + add + commit to git
- delete second file:
  rm+status+add+commit
- create the file again, add, commit, then delete again, this time using git rm
- create a file, add, commit, then rename + add + commit
  mv …
  status+add+commit
- do it again, this time using git mv
- git log
- git log --oneline
- git log -p
- git status
- git diff
- git diff --staged
- git restore: change a file and restore it (before staging)

## Merge Exercise

(the commands we used can be found on the end)

- 0. make sure you have at least one commit (if you are not sure if you have any commit, use git log), with any content you like.
- 1. Simple Merges:
  - Assumption: You have some commits in your repository, all committed to "master"; You are on "master".
  - Instructions:
    - 1.1.1 Create a new branch "food" **from branch "master"** and switch to it.
      - Use 'git log'. You should see both "master" and "food" on the same commit.
    - 1.1.2 Add a "cookies.txt" file with a recipe to this branch. commit.
      - Use 'git log --graph' to see your work. The graph should appear as one vertical line, with no branching.
    - 1.1.3 Make some changes to "cookies.txt" and commit again. Repeat this several times.
      - Use 'git log --graph' to see your work. The graph should still appear as one vertical line.
    - 1.2.1 Create a new branch "music" **from branch "master"**. switch to it.
      - Use 'git log --graph' to see your work. You should **not** see the commits from the "cookies" branch.
      - Use 'git log --graph --all' to see your work. You should see the commits from both branches.

        - 1.2.2 Create a "song.txt" file, containing the lyrics of a song you like. Commit it.
        - 1.2.3 Add a new line to your song.txt file. Commit the change.
            - Use 'git log --graph' to see your work. You should **not** see the commits from "cookies".
            - Use 'git log --graph --all' to see your work. You should see the commits from all branches.
        - 1.3.1 Create a new branch "fun" **from the master branch**. Switch to it.
            - git log --graph
            - git log --graph --all
        - 1.3.2 Create a "jokes.txt" file, with a few jokes. Commit it.
        - 1.3.3 Add a few more jokes to "jokes.txt" and commit again.
            - git log --graph
            - git log --graph --all
        - 1.4.1 Time to merge all your work to the master branch!
            - Switch to the master branch.
        - 1.4.2 Use `git merge food` to merge your cookie recipe. Make sure to carefully read the output of the merge command.
            - What happened?
                - git log --graph --all
        - 1.4.3 We still need to merge in our music.
            - Use `git merge music` to merge your favorite song.
            - What happened?
                - git log --graph --all
        - 1.4.4 And let's not forget to bring in the fun.
            - Merge in the "fun" branch as well.
            - git log --graph --all
        - 1.4.5 Time to clean up!
            - Bonus: Use "git branch --merged" and "git branch -v --merged" to see the merged branches.
            - Delete merged branch with "git branch -d nameofbranch"
            - enjoy your work
                - git log --graph --all

- 2. Merging just works!
    - In this exercise we will change the same file in two different places. git will merge the changes without conflicts.
    - Notes:
        - Although it is usually better not to work directly on the "master" branch we will do it to expedite this exercise.
        - For this exercise, please work cleanly - changing the file in two **different** places as instructed. We will tackle merge conflicts a little later.
    - Assumption: You have some commits in your repository, all committed (or merged) to "master". You are on the "master" branch.
    - Instructions

- 2.1.1 Create a new file "star.py" with the content below:

```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
    forward(200)
    left(160)
    if abs(pos()) < 1:
        break
end_fill()
exitonclick()
```

Bonus: run ‘python star.py’ (‘python3’ instead of ‘python’ if you work on a mac)

- 2.1.2 Add and commit the code to the master branch, with the message "added star". use git log and see the new commit.

- 2.2 Let's change some colors.
  - 2.2.1 Create and switch to a new branch called "**blue_star**".
  - 2.2.2 Modify the 2nd line in the file to `color('green', 'blue')` (or another color if you want)
    Use `python star.py` to run the file and enjoy your colorful creation.
  - 2.2.3 Add and commit the code. Make sure it’s committed (and on the right branch) - “git log --graph”.

- 2.3 Meanwhile - someone else has committed some changes. Let's simulate this on the master branch:
  - 2.3.1 Switch to the master branch.
  - 2.3.2 Run python star.py to see the star in the original colors
  - 2.3.3 Running `git log` **shouldn’t** show the "blue_star" branch commits.
  - 2.3.4 Let's make the star a little more spiky.
    Change `left(160)` to `left(170)`
    Use `python star.py` to see it :-)
  - 2.3.5 Commit your work with the message "spiky!!".
  - 2.3.6 git log --graph --all, to see the current state.

- 2.4 Let's merge in the colors!
    - Run `git branch` to see the list of branches and the current ("checked out") branch marked with an asterisk.
    - 2.4.1 (still on the master branch):
      run `git merge blue_star` to merge the work from the other branch
      (Accept the "merge" commit message)
    - 2.4.2 Use `cat star.py` to see the merged code. Use `python star.py` to see the merged result.
        - Use `git log --graph --all --oneline` to see the log.
    - 2.4.3 Run `git show -m` to see the diff in reference to the first AND then the second parent.
    - 2.4.4 Run `git show -c` to see a COMBINED merge. (notice the + and - are in two different columns, @@@ at top)
- 2.5 Cleanup
    - 2.5.1 delete the merged branch.
      Use git log --graph --all to enjoy. Celebrate your work.

- *2.6 Bonus: Try this trick again if you want - work in two different parts of your code, modify different files and parts of the code in different branches and merge the work later. As long as you don't change the exact same lines - git will probably easily merge your work. Stuff to try:*
    - *change the drawing speed -- delay(5)*
    - *change the shape -- shape("turtle")*
    - *change the pen width -- width(8)*
    - *change the pen colors while drawing -- pencolor([random.random() for i in range(3)])*
    - *draw a circle(150) around the star…*
    - *draw more stars…*

Commands we used:

- git show <commit hash>
- git show <branch name>
- git show HEAD
- git branch <name of created branch>
- git switch <branch to switch to>
- git switch -c <name of created branch> <name of branch to fork from>
- git switch -c <name of created branch> (forking from current branch)
- git branch
- git branch -v
- git branch -d
- git branch -D
- git log –all
- git log –graph
- git log –oneline –graph –all
- git merge <branch to merge to current branch>

Useful resources:

- Git book

# conflict resolution exercise

Commands we used:

- git merge
- git merge –abort
- git add
- git commit

*Handling Conflicts*

- *This time we are going to create conflicting branches.*
- *3.1 Work on a new "large star" feature branch:*
  - *3.1.1 Create and switch to a new branch "large_star" from main.*
  - *3.1.2 Add* `speed(10)` *just below the import command (just below the first line)*
  - *3.1.3 Change the* `forward(200)` *command to* `forward(300)` *. Run python star.py*
  - *3.1.4 Commit your changes*
- *3.2 Work on a conflicting feature (To save time, we work directly on main):*
  - *3.2.1 Switch back to the main branch.*
  - *3.2.2 Run python star.py to see the previous version of your work.*
  - *3.2.3 Modify the size of the star:* `forward(50)`
    *3.2.4 A line below “end_fill()”, add:*
    `write("May the force be with you!", align="right")`
  - *3.2.4 Run python star.py*
  - *3.2.5 Commit your changes.*
- *3.3: Time to merge*
  - *3.3.0 run 'git status'. Make sure your working copy is clean.*
    *3.3.1 run 'git log --graph --all'.*
  - *3.3.2 Merge the other branch*
    *Read the message carefully :-)*
  - *3.3.3 run git status*
  - *3.3.4 run git diff*
  - *3.3.5 Open the file with the conflicts in your editor.*
  - *3.3.6 Resolve your conflicts by choosing the implementation you like.*
    *Remove the marker lines.*
  - *3.3.7 git add your file to mark it as resolved*
  - *3.3.8 git status*
  - *3.3.9 git add and commit*
  - *3.3.10 git log --graph*
  - *Bonus: git log --graph -p -c*

- *3.4 Bonus: Try this again, adding more code in two different branches, with more conflicts, and try to resolve them.*

**Handling Conflicts Using the GUI**

- Resolve the conflicts from exercise 3 (or other conflicts, whatever you prefer) using PyCharm ONLY.
    - PyCharm (and other IDEs) has a special tool for resolving conflicts, give it a try!

# git exercise - upstream

(the commands we used can be found on the end of the file)

**Part 1**

- Show your public key:
  `cat ~/.ssh/id_rsa.pub`

    - If the public key does not exist, create a key pair:
      `ssh-keygen`
      `(The terminal will ask you some questions. Don't answer the questions. Don't type anything. Just hit enter enter enter)`

- Login to www.github.com and Add your ssh key here:
  https://www.github.com/settings/keys
  (unless it's already there if you used it before)

- Test your SSH connection:
  `ssh -T git@github.com`
  (Enter "yes" for host confirmation if needed)

**Part 2**

1. Using the web interface, create a project on www.github.com (Without a README)
    - Visibility level: Choose "Private"
2. Under "Quick setup — if you've done this kind of thing before" - click "SSH".
3. Copy the SSH url
    - `git remote add origin <SSH url>.`
4. View your configured remotes:
    - `git remote`
    - `git remote -v`
5. Push your work and set the upstream tracking branch:
    - `git push -u origin main`
6. Observe your local remote branches:
    - `git log --graph`
    - `git branch`
    - `git branch --remote`
    - `git branch --all`

    - `git branch -vv`
7. Refresh your web browser to see all the commits.
    - In your project, click on some files (or folder) to view their content.
    - Hit "commits" to view the commit log
    - Click on one commit to view the commit details
    - When viewing one file:
        - Click [Blame] to view the file, with all lines annotated by the commit and author.
        - Click [History] to view the history of this file.
        - From this history view, click on a commit to see the full commit details (all files in the commit).
            - Try both “split” and “unified” view.

**Part 3**

7. On your computer, create a new file: **ascii_art.txt** and add a nice drawing using your editor: https://www.asciiart.eu/
    - Add and commit the file
8. Run a few commands to see the new commit.  It is not in your Github repo yet.
    - `git log --graph`
    - `git branch -vv`
9. Push your commits using "git push"
    - Refresh your web browser to see the new commits online.
    - View the commit log and the ascii_art.txt file contents online

**Part 4**

10. Open Github's Web IDE
    - Edit your ascii_art.txt file online: add a large ascii art title using figlet online: https://doodlenerd.com/web-tool/figlet-generator
    - Add another header to song.txt (or another text file).
    - Commit the changes to **main** / **master** branch with a short comment ("added ascii art titles")
11. Exit the web IDE and view your changes online (through github's interface)
12. Fetch your changes
    - Use "`git fetch`" to fetch the changes you made online.
    - Run "`git status`"
    - Run "`git log --all --graph`" to see them
    - Run "`git branch -vv`"
    - View your files using **cat ascii_art.txt** . They don't contain the titles yet
13. Merge your changes by performing a fast-forward:
    - Run "`git merge --ff-only`"
      (This is a shortcut to
      `git merge origin/main --ff-only`
      Since main is tracking origin/main)
14. View your changes:
    - Run "`git status`"
    - Run "`git log --all --graph`" to see them

- Run "`git branch -vv`" to
- View your files using **cat ascii_art.txt**

**Part 5**

15. On the Web IDE, add a **README.md** file. Use some markdown to add something like this:

````
# My Cool Project

This is just a sample project to learn **git**.

Stuff to try with this code:

```bash
python star.py
```

A few good resources:

* [The git book](https://git-scm.com/book/en/v2)
* [Markdown Cheat
Sheet](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet)
* [git extras](https://github.com/tj/git-extras)

A checklist:

- [x] Learn git
- [x] Become a master
- [ ] Profit!

![Oh
Yeah!](https://c.tenor.com/vkpCnKNg7UkAAAAM/mind-blown-amazed.gif)
````

    Commit your work and view your changes online on the project page.

16. *(Create a few more small commits please - in different files - it will help us in the next exercises.)*

17. Meanwhile, without fetching, modify one or more files locally (on your machine) and create a new commit.
    - Note: **main** on your machine and **main** online on github are different!

18. Do the merge:

    - Option A: using fetch+merge:
    - `git fetch`
    - `git log --graph --all`
    - `git merge --ff-only`
      `(This will fail!)`
    - `git merge`
    - `git log --graph --all`
    - Option B: using pull:
    - `git pull --ff-only`
      `(This will fail!)`
    - `git pull`
    - `git log --graph --all`
19. Update the remote repo:
    - `git push`
    - `git log --graph --all`

**Part 6**

20. Check the status of your local repository:
    - Make sure all files are tracked and committed (git status)
    - Make sure everything is pushed
21. Delete your local repository.
22. Clone it from the remote repository with git clone (Using the ssh url).

Good luck!

Commands we used:

- git remote
- git remote add origin <url>  (the name “origin” is only a convention)
- git remote -v
- git push origin master
- git push -u origin master
- git push
- git branch –all
- git branch -vv
- git branch –remotes
- git fetch
- git pull ( = fetch + merge)
- git clone <url>

# git stash exercise

Commands we used:

- git stash (git stash push)
- git stash pop
- git stash list
- git stash list -p
- git stash -m "some message"
- git stash drop

**exercise**

1 start a new git repository (mkdir, cd, git init)
2 create a file “emoji.py”, type **print('\N{balloon}')** in it. Add and commit.
3 python emoji.py (mac users: python3 emoji.py)

4 start another branch and switch to it
5 change the word ‘balloon’ to ‘snake’. Add and commit
6 python emoji.py (mac users: python3 emoji.py)

7 switch back to the master/main branch
8 change the word ‘balloon’ to ‘snail’ (don’t add/commit)
9 python emoji.py (mac users: python3 emoji.py)

10 git switch to the other branch (it should fail. pay attention to the output)
11 type “git branch”. the branch didn’t switch! Why?
12 python emoji.py (mac users: python3 emoji.py)

13 stash, then (try again to) switch to the other branch.
14 python emoji.py (mac users: python3 emoji.py)
15 type “git stash list”, to see your new stash entry.

16 switch back to the master/main branch.
17 python emoji.py (mac users: python3 emoji.py)

18 bring the stashed content back, using “git stash pop”
19 python emoji.py (mac users: python3 emoji.py)
20 add and commit to master

# rainbow exercise

https://github.com/zkmoty/devboostjan25rainbow

1. clone (git clone + cd)
2. Every team conducts a short face-to-face meeting. **During the meeting**:
    - 1.1 They choose a team name/
    - 1.2 Together on one PC, they:
        - Create a new branch from main, called "team1" / "team2" / etc.. +
        - On this branch edit **index.html** and change only the line "Team x: …" and modify "..." to your team name.
        - This can be done via:
            - Option A: Through the web interface (create branch + edit the file **in the correct branch**)
            - Option B: On a developer machine (Don't forget to add, commit and push the new branch).
        - Make sure you can view the branch through Github's web interface.
    - 1.3 Choose a "team leader". The team leader creates a new issue. The issue contains:
        - Subject: "Contributions by Team *{The team name}*"
        - Body:
            - A checklist with names of users + what they are going to add to the gallery. For example:

```
- [ ] Left card: @first_user: Monkeys dancing macarena
- [ ] Center card: @other_user: Unicorns singing karaoke
- [ ] Right card: @third_user: Cats eating ramen
```






- (Also: Decide who is code reviewing who (In a circular fashion - 1st member > 2nd member > 3rd member > 1st member). This can be written in the issue as well.

2. After the meeting, each member:
    - Clones the project to their machine.
    - Switch to your team's branch on your machine, and view the index.html page in the browser (with your team name):
      `git switch team_x`
      This creates a local branch team_x tracking the remote branch origin/team_x, and switches to it.
    - Creates a new branch **from their team branch** called **teamX_name,** for example **team3_moty** and switches to work on it.
    - In the repository, they create a **pages/my-name.html** file (for example: **pages/moty-zk.html** file) and add there a nice html file with either a song, a travel recommendation, a recipe, a good story or a joke, a funny picture or a meme - anything they would like to add.
        - Please try to include a typo or some kind of small mistake in your page….
    - In **index.html** each member is editing their card:
        - Adding a nice image (either by using a URL or by adding it as a file to the **images/** folder)
        - Adding a small sentence - a nice quote or anything they want.
        - Adding a link like this: **`<a href="pages/my-name.html">Link Text</a>`**
    - Committing the changes:
        - The comment should include "Refs #123" where 123 is the number of the created issue.
    - Pushing their branch to github.
    - Creating a pull request from **teamX_name** to **teamX (!)**
        - Write down what was done.
        - Assign the pull request (PR) to your reviewer.
    - Hits the corresponding checkbox in the issue page 

3. Each member code reviews another member's Pull Request (PR), adding comments as needed.
    - Pair programming can be used to go over the code, resolving all issues, committing at least one more commit to each member's branch.
    - Once PRs are approved - merge them by:
        - Option A: via the web interface
        - Option B: by merging them on the machine and pushing them online.

4. When all branches are merged into the team's branch:
    - Every member should checkout and pull the team branch on their PC.

## git exercise - cherry picking

- Take a look at this project:
  https://github.com/zkmoty/huge
- and the graph/network page:
  https://github.com/zkmoty/huge/network
  It shows **main**, **release_1** and **release_2** (for zkmoty)
  you can also view this using 'git log --oneline --graph --all'

- fork via github web interface
- clone your fork to your machine
- cherry pick beatles stuff from main to release_2
- cherry pick all house drawing stuff from main to release_1  (Optional: using pycharm)

commands we used:
git cherry-pick <cmt id> <cmt id> <cmt id> …

git ecosystem example links:
https://www.producthunt.com/products/git-story#git-story
https://github.com/Bhupesh-V/ugit
https://github.com/initialcommit-com/git-sim

merge strategies elaboration:
https://git-scm.com/docs/merge-strategies#Documentation/merge-strategies.txt-ort

command line arguments vs input
https://stackoverflow.com/questions/53571637/command-line-arguments-vs-input-whats-the-difference


Daniel
@growing_daniel
I met my wife in a github issue thread
6:51 AM · 11 Jan 23
1.1M Views 536 Retweets 184 Quote Tweets
10.3K Likes
Mickey Friedman @mickeyxfrie... · 16h
Replying to @growing_daniel
glad you found a girl who could commit
28.1K 35 87 2,098
Show replies
Adam Lyttle @adamlyttleapps · 9h
Replying to @growing_daniel
Glad you two merged
I'll see myself out
9,026 1 164