

Advanced git techniques

Johannes Holke

For with Git nothing shall be impossible.
- Luke 1:37



Knowledge for Tomorrow



Intro

In this talk I present some advanced techniques of git that I learned during my years working with git.

I am an (expert) git **user**, I am **not** a **git expert** – so don't expect me to know everything.

I will show you specific use cases, not detailed instructions.



What you need

To repeat the exercises:

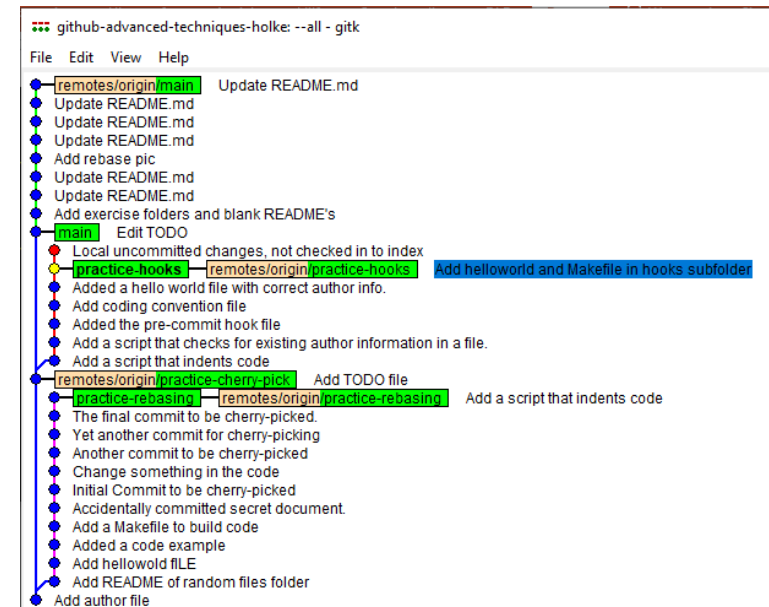
1. **Fork** the repository <https://www.github.com/holke/git-advanced-workshop>
2. Clone your fork
3. Follow the exercises/*/README.md

Terminal:



```
MINGW64/c/Users/holk_jo
holk_jo@SC-... MINGW64 ~
$
```

gitk:



```
github-advanced-techniques-holke: --all - gitk
File Edit View Help
-remotes/origin/main Update README.md
- Update README.md
- Update README.md
- Update README.md
- Update README.md
- Add rebase pic
- Update README.md
- Update README.md
- Add exercise folders and blank README's
- main Edit TODO
- Local uncommitted changes, not checked in to index
- practice-hooks -remotes/origin/practice-hooks Add helloworld and Makefile in hooks subfolder
- Added a hello world file with correct author info.
- Add coding convention file
- Added the pre-commit hook file
- Add a script that checks for existing author information in a file.
- Add a script that indents code
-remotes/origin/practice-cherry-pick Add TODO file
- practice-rebasing -remotes/origin/practice-rebasing Add a script that indents code
- The final commit to be cherry-picked.
- Yet another commit for cherry-picking
- Another commit to be cherry-picked
- Change something in the code
- Initial Commit to be cherry-picked
- Accidentally committed secret document.
- Add a Makefile to build code
- Added a code example
- Add helloworld FILE
- Add README of random files folder
- Add author file
```

Content

- Cherry-picking (mild warmup)
- Interactive rebase (rewrite history)
- Reflog (you can't mess up)
- Hooks (automation 1)
- Filters (automation 2)
- Additional worktrees
- Bisect



Cherry-picking

Apply commits from other branches without merging the whole branch.

Remember:

```
git cherry-pick <commit>
```

```
git cherry-pick <commitA>^..<commitB>
```



Interactive rebase

Rewrite your history. Unleash the full power of git.



WARNING: If you force push rewritten history into branches that other people work on, you will make enemies for life. Only use interactive rebasing if you are certain that you are the only person working on that branch.



Interactive rebase

Rewrite your history. Unleash the full power of git.

Remember:

```
git rebase -i HEAD~N
```

Workflow to edit a commit:

1. `git reset --soft HEAD~`
2. EDIT COMMIT
3. `git commit -c ORIG_HEAD`



Reflog



Remember:

git reflog



Hooks

A hook is a script that is automatically executed during a git action (commit, push, etc.).
A hook can decide whether to abort this action.

Enforce coding conventions

Deny commits that do not compile

...

Remember:

`.git/hooks/`

Useful:

`for file in `git diff --cached --name-only``



Filter

We have all been here:

```
void merge(int arr[], int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    cout << "No segfault so far...\n"; // debug

    for(int i = 0; i < n1; i++) {
        /* I think it breaks here, but am not sure */
        cout << "Before: " << L[i] << std::endl; // debug
        L[i] = arr[l + i + 1];
        cout << "After: " << L[i] << std::endl; // debug
    }
    for(int j = 0; j < n2; j++) {
        /* I am debugging this for 3 hours now.
        * God, why j, m, R...?!?!?!?!
        * Give your variables useful names, arghhhh!
        */
        R[j] = arr[m + 1 + j];
    }

    cout << "If we reach this without segfault, we should be good\n"; // debug
}
```



Filter

With filters, we can modify file contents on-the-fly before we commit them or check them out.

Keep your private code bites out of the repo.

- print-debugging
- comments that are not meant for others

clean: modify before git sees it

smudge: modify before i see it (rarely used)

Remember: .git/config

```
[filter "gitignore"]
clean = exercises/filter/gitignore_filter.scp
smudge = cat
```

.git/info/attributes

```
*.c filter=gitignore
*.h filter=gitignore
*.py filter=gitignore
```



Worktrees

With worktrees we can make additional copies of our working directory.

- No copying of .git files
- git prevents you from working on the same branch twice

Why?

- Code review
- Work on multiple features
- Work on branch A while branch B compiles
- ...

Remember:

```
git worktree add PATH BRANCH
```

```
git worktree prune
```



Bisect

Binary search your history for bad commits.

Remember:

```
git bisect start
git bisect bad
git checkout GOOD_COMMIT
git bisect good
# Mark incoming commits with git bisect good/bad
# git will tell you the first bad commit
git bisect reset
```



Thank you for your attention

