Usage

- Basics
- Pushing / Merging Branches
- Submodules

Basics

First, check out this brief explanation on what Git is, why it was created, and general descriptions of features or ideas Git is built around. This will help a lot to understand the commands you are using, instead of just searching for a command that does something favorable. You will gain a lot of context by reading this page.

If you are new to git entirely, I'd recommend checking out this interactive tutorial to learn git branching and running through the examples they host there. This will get you working with Git quickly in various situations and difficulties. After this, you will have some experience with Git!

The Git-scm Book is a good read and also serves as a great online reference. Once you have an idea of what you are loooking for and where you need further your understanding, this will be useful to you.

And when all else fails, ohshitgit outlines what to do in a few `oh shit, I fucked up` scenarios. dangitgit is the same reference, without the bad language and thus is more suitable to leave up on a work monitor or to share in a presentation.

Create a Repository

To create a repository, just create a directory or enter the root directory of the project you want to turn into a repository, and type `git init`. This initializes the directory as a *local* repository. To add the repository to GitHub and track it remotely, you'll need to login to GitHub and click 'New Repository', name the repository the same as your root folder and continue. GitHub will provide you with the rest of the instructions, but for completeness, tweak the lines below to push your local repository to your new remote on GitHub -

```
git remote add origin git@github.com:<username>/<reponame>.git
git push -u origin master
```

We just created the `origin` remote. This is the remote that is displayed and tracked on GitHub, when you clone your repository you are on a local remote, which means until you `git push <remote> <branch>` your changes will only be saved and tracked on your local machine.

Using a remote via SSH such as the above `git@github.com:<username>/<reponame>.git` requires you have configured an SSH key with your GitHub account that is associated with the machine you are pushing from. If you haven't already, check out Creating SSH Login Keys and simply `cat ~/.ssh/<USERKEY>.pub` the public key of your user and copy it over into your GitHub settings.

If you'd rather not mess with things like this, see how to create a [Person Access Token](#) below.

Ignoring Files

Within a Git repository, `.gitignore` files can be seen specifying a list of files or directories that Git should ignore when tracking changes. For example, this is useful when a project is expected to contain build files generated after being cloned. We would not want the user to then make a commit publishing the files they generated when building the project for their system, we would want to provide a clean slate for the next person that clones the project.

After creating a `.gitignore` file, the syntax below can be followed to specify files and directories to be ignored.

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

> " [Git-scm - Ignoring Files](#)

Check out [GitHub - gitignore repo](#) for some templates used in popular languages, like C, C++, or Python.

If you are ignoring a file that Git has already previously tracked, it may be necessary to remove the file (or directory) from Git's cache using the command below

```
# Remove cached file
git rm --cached path/to/file
# Remove cached directory
git rm -r --cached path/to/
# Remove all cached files
git rm -r --cached .
```

Commit Guidelines

Before you commit, run `git diff --check`, which identifies possible whitespace errors and lists them for you.

Write your commit message in the imperative: "Fix bug" and not "Fixed bug" or "Fixes bug." This convention matches up with commit messages generated by commands like git merge and git revert.

Use `git add --patch` to partially stage files (covered in detail in https://git-scm.com/book/en/v2/Git-Tools-Interactive-Staging#_interactive_staging)

The project snapshot at the tip of the branch is identical whether you do one commit or five, as long as all the changes are added at some point

As a general rule, your commit messages should start with a single line that's no more than about 50 characters and that describes the changeset concisely, followed by a blank line, followed by a more detailed explanation

`git log --pretty=oneline` shows a terse history mapping containing the commit id and the summary

If the config option merge.summary is set, the summaries from all merged commits will make their way into the merge commit message

> `git pull --rebase` What's happening here? Git will rewind (undo) all of your local commits, pull down the remote commits then replay your local commits on top of the newly pulled remote commits. If any conflicts arise that git can't handle you'll be given the opportunity to manually merge the commits then simply run `git rebase --continue` to carry on replaying your local commits.
>
> How to Avoid Merge Commits in Git - Kernowsoul

`git shortlog` uses summary lines in the changelog-like output it produces - `git format-patch`, `git send-email`, and related tools use it as the subject for emails.

Reflogs, a local history accessible with `git reflog`, is intended to help you recover from stupid mistakes by providing the hashes along with output similar to `git shortlog`.

Reversing Changes

Use `git reset <remote>` for local changes -

```
# one commit in the past
git reset HEAD^


# 2 commits in the past
git reset HEAD^^


# 2 commits in the past
git reset HEAD~2


# 3 commits in the past
git reset HEAD~3
```

Use `git revert <remote>` for changes that have already been pushed to a remote -

```
# 1 commit in the past
git revert HEAD


# 2 commit in the past
git revert HEAD^


# 3 commits in the past
git revert HEAD^^


# 3 commits in the past
git revert HEAD~2


# 4 commits in the past
git revert HEAD~3


# Note that HEAD could be replaced with v0.2, or any active branch that exists on the remote
```

Modifying Previous Commits

WIP

For now, [Here's a good tutorial](#)

:)

```
git rebase --interactive --autosquash --rebase-merges --root master
```

Pushing / Merging Branches

Pushing

If we run ... `git push <remote> serverfix`
Git automatically expands the `serverfix` branchname out to `refs/heads/serverfix:refs/heads/serverfix` ,
where the sytax is `local:remote` ..

You can use this same syntax when pusshing a local branch into a remote branch that is named
differently. If you didn't want it to be called `serverfix` on the remote, you could instead run `git push origin serverfix:awesomebranch` to push your local `serverfix` branch to the `awesomebranch` branch on the
remote project.

Should you see the errors below when attempting to push, see the Pull / Merge > Resolving
Conflicts section of this page for steps on merging your branches, resolving the conflicts, and then
completing your push.

```
git push origin v0.2


To github.com:shaunrd0/CMake.git

! [rejected]        v0.2 -> v0.2 (non-fast-forward)

error: failed to push some refs to 'git@github.com:shaunrd0/CMake.git'

hint: Updates were rejected because the tip of your current branch is behind

hint: its remote counterpart. Integrate the remote changes (e.g.

hint: 'git pull ...') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Shared Remote Server Workflow

*You should also be able to share your branches by pushing them to a shared server, working with
others on shared branches and rebasing your branches before they are shared. This being said, it's
possible to have a workflow where each developer has write access to their own public repository
and read access to everyone else's -

1. The project maintainer pushes to their public repository.
2. A contributor clones that repository and makes changes.
3. The contributor pushes to their own public copy.
4. The contributor sends the maintainer an email asking them to pull changes.
5. The maintainer adds the contributor's repository as a remote and merges locally.
6. The maintainer pushes merged changes to the main repository.

Branching

Basic git branch commands -

Checkout and create new branch if it doesnt exist

```
git checkout -b branchname
```

`git log --all --graph --decorate --oneline --simplify-by-decoration` will output your history in a format similar to the Network Graph on GitHub

```
git log --all --graph --decorate --oneline --simplify-by-decoration

* 8221652 (HEAD -> master, origin/master, origin/HEAD) merge v0.4 into master
* 27e6e1c (origin/v0.4, v0.4) Fix for tab spacing in vim
| * feb1da1 (refs/stash) WIP on master: 807e0b3 Reorganized C problem 4
|/
* 807e0b3 (origin/v0.3) Reorganized C problem 4
* 2fc4266 (origin/v0.2) Finishing up v0.2
| * 9187276 (origin/v0.1) Cleaned up the README.
|/
* f1b858b Initial commit of first CMake project
```

The output can be formatted further, and linked with aliases within git -

```
git config --global alias.lg "log --all --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"
```

There are also other simpler options for similar output with less information -

```
git show-branch
git show-branch --all
```

Alternatively, if you would rather a GUI - run `gitk` - provided it's configured correctly.

To delete a branch, local or remote, see the commands below -

Remove a local branch

```
git branch -d the_local_branch
```

Remove a remote branch. Be careful with this command! Be sure you know that you want to delete the branch forever.

```
git push origin :the_remote_branch
git push origin --delete the_remote_branch
```

Pull / Merge

Before merging, commit and push a 'checkpoint' to your version or feature branch. If you do not, git will squash the history from your branch into master - this commit can serve as a reference for changes merged into master later on. Should you forget to do this, the merge could still be traced with more effort.

merge `master` into the `test` first to resolve any conflicts on the `test` branch itself. After the `test` branch is clean, up-to-date, and pushed to origin, I'll `git checkout master` and `git merge test`.

`git merge origin/master`. If you want to fast-forward, run `git merge --ff-only origin/master`

The `--squash` option takes all the work on the merged branch and squashes it into one changeset producing the repository state as if a real merge happened, without actually making a merge commit.

Also the `--no-commit` option can be useful to delay the merge commit in case of the default merge process.

Resolving Conflicts

Problems pushing your local changes to a remote (origin) ?
`git pull <remote> <branch>` and resolve the conflicts by following the instructions below .

When attempting to pull or merge branches, there can sometimes be new changes to the same content within the same files on the two different branches. Since git wants to be sure that you retain the changes you want, it pauses our merge and prompts us to resolve these conflicts before creating a final commit to finish our merge.

```
git pull origin v0.2

From github.com:shaunrd0/CMake
* branch          v0.2      -> FETCH_HEAD
Auto-merging 4-Ch2-course-laucher/CMakeLists.txt
CONFLICT (content): Merge conflict in 4-Ch2-course-laucher/CMakeLists.txt
CONFLICT (add/add): Merge conflict in 4-Ch2-course-laucher/4-problems/CMakeLists.txt
Auto-merging 4-Ch2-course-laucher/4-problems/CMakeLists.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Below, we can see that my branch has conflicts with `4-Ch2-course-launcher/CMakeLists.txt` - So, to resolve these, we would run `vim 4-Ch2-course-launcher/CMakeLists.txt`

```
On branch v0.2
You have unmerged paths.
   (fix conflicts and run "git commit")
   (use "git merge --abort" to abort the merge)


Unmerged paths:
   (use "git add <file>..." to mark resolution)


   added:      4-Ch2-course-laucher/4-problems/CMakeLists.txt
 both modified:   4-Ch2-course-laucher/CMakeLists.txt


no changes added to commit (use "git add" and/or "git commit -a")
```

`vim <path/to/conflict/file>` and you will notice syntax similar to the below has been added to your file -

```
Some text in a file that has no conflict.


<<<<<<< HEAD
Some text that was changed on the local HEAD.
=======
Some text that was also changed on the remote we are attempting to merge with
>>>>>>> feature-branch


Some more text in a file that has no conflict.
```

All that Git is asking us to do here is delete the changes that we don't wish to keep, and then `git commit -m "Commit message"` to complete our merge. If you want to abort the merge, run `git status` to see how, or just run `git merge --abort`.

So, in this case if we wish to keep the changes that are on our local `HEAD`, and overwrite the changes on our `feature-branch`. Just modify the file, deleting all of the added syntax from the merge conflicts described by git, and any changes that may go with them -

```
Some text in a file that has no conflict.


Some text that was changed on the local HEAD.


Some more text in a file that has no conflict.
```

Our merge conflict is resolved. check `git status`, stage your changes with `git add` and make the commit to finish the merge.

Submodules

Submodules are a useful feature of git, and allow us to nest projects within our own project. A good example of a project that would take advantage of this is a dotfiles repository - mine can be found on GitLab and has several submodules.

I've recently been working on kot, a linux dotfiles CLI tool, and I have been testing it with my dotfiles repository within a docker container. I thought this was a good chance to write down some useful commands for dealing with git submodules.

If you want a repository to test these commands, feel free to clone kot and play around with the submodules within.

```
git clone https://gitlab.com/shaunrd0/kot
cd kot
```

To show a status of all submodules -

```
git submodule status --recursive

-7877117d5bd413ecf35c86efb4514742d8136843 dot (heads/master)
-826d5691ac7d36589591314621047b1b9d89ed34 dot/.vim/bundle/Colorizer
-3ea887d2f4d43dd55d81213517344226f6399ed6 dot/.vim/bundle/ale
-293a1062274a06be61797612034bd8d87851406e dot/.vim/bundle/clang_complete
-d80e8e2c1fa08607fa34c0ca5f1b66d8a906c5ef dot/.vim/bundle/supertab
-afb8db4f81580771c39967e89bc5772e72b9018e dot/.vim/bundle/unicode.vim
-cb1bc19064d3762e4e08103afb37a246b797d902 dot/.vim/bundle/vim-airline
-d148d42d9caf331ff08b6cae683d5b210003cde7 dot/.vim/bundle/vim-airline-themes
-b2a0450e23c63b75bbeabf4f0c28f9b4b2480689 dot/.vim/bundle/vim-signify
```

So I have `dot` as a submodule, and `dot` also has several submodules we inherit with the `--recursive` option. The same command without recursion is shown below

```
git submodule status

-7877117d5bd413ecf35c86efb4514742d8136843 dot (heads/master)
```

Notice in all of the output below, there is a minus sign `-` before each submodule. This means they have not been initialized. To summarize the meaning of this output, I'll quote from `man git-`

`submodule`

> 66 This will print the SHA-1 of the currently checked out commit for each submodule, along with the submodule path and the output of git describe for the SHA-1. Each SHA-1 will possibly be prefixed with - if the submodule is not initialized, + if the currently checked out submodule commit does not match the SHA-1 found in the index of the containing repository and U if the submodule has merge conflicts.

To update and initialize all submodules recursively, we can use the following command

```
git submodule update --init --recursive


Submodule 'dotfiles/dot' (https://gitlab.com/shaunrd0/dot) registered for path 'dotfiles/dot'
Cloning into '/home/kapper/Code/kotd/dotfiles/dot'...
warning: redirecting to https://gitlab.com/shaunrd0/dot.git/
Submodule path 'dotfiles/dot': checked out '7877117d5bd413ecf35c86efb4514742d8136843'
Submodule '.vim/bundle/Colorizer' (https://github.com/chrisbra/Colorizer) registered for path
'dotfiles/dot/.vim/bundle/Colorizer'
Submodule '.vim/bundle/ale' (https://github.com/dense-analysis/ale) registered for path
'dotfiles/dot/.vim/bundle/ale'
Submodule '.vim/bundle/clang_complete' (https://github.com/xavierd/clang_complete) registered for path
'dotfiles/dot/.vim/bundle/clang_complete'
Submodule '.vim/bundle/supertab' (https://github.com/ervandew/supertab) registered for path
'dotfiles/dot/.vim/bundle/supertab'
Submodule '.vim/bundle/unicode.vim' (https://github.com/chrisbra/unicode.vim) registered for path
'dotfiles/dot/.vim/bundle/unicode.vim'
Submodule '.vim/bundle/vim-airline' (https://github.com/vim-airline/vim-airline) registered for path
'dotfiles/dot/.vim/bundle/vim-airline'
Submodule '.vim/bundle/vim-airline-themes' (https://github.com/vim-airline/vim-airline-themes) registered for
path 'dotfiles/dot/.vim/bundle/vim-airline-themes'
Submodule '.vim/bundle/vim-signify' (https://github.com/mhinz/vim-signify) registered for path
'dotfiles/dot/.vim/bundle/vim-signify'
Cloning into '/home/kapper/Code/kotd/dotfiles/dot/.vim/bundle/Colorizer'...
Cloning into '/home/kapper/Code/kotd/dotfiles/dot/.vim/bundle/ale'...
Cloning into '/home/kapper/Code/kotd/dotfiles/dot/.vim/bundle/clang_complete'...
Cloning into '/home/kapper/Code/kotd/dotfiles/dot/.vim/bundle/supertab'...
Cloning into '/home/kapper/Code/kotd/dotfiles/dot/.vim/bundle/unicode.vim'...
Cloning into '/home/kapper/Code/kotd/dotfiles/dot/.vim/bundle/vim-airline'...
Cloning into '/home/kapper/Code/kotd/dotfiles/dot/.vim/bundle/vim-airline-themes'...
```

```
Cloning into '/home/kapper/Code/kotd/dotfiles/dot/.vim/bundle/vim-signify'...
Submodule path 'dotfiles/dot/.vim/bundle/Colorizer': checked out
'826d5691ac7d36589591314621047b1b9d89ed34'
Submodule path 'dotfiles/dot/.vim/bundle/ale': checked out '3ea887d2f4d43dd55d81213517344226f6399ed6'
Submodule path 'dotfiles/dot/.vim/bundle/clang_complete': checked out
'293a1062274a06be61797612034bd8d87851406e'
Submodule path 'dotfiles/dot/.vim/bundle/supertab': checked out
'd80e8e2c1fa08607fa34c0ca5f1b66d8a906c5ef'
Submodule path 'dotfiles/dot/.vim/bundle/unicode.vim': checked out
'afb8db4f81580771c39967e89bc5772e72b9018e'
Submodule path 'dotfiles/dot/.vim/bundle/vim-airline': checked out
'cb1bc19064d3762e4e08103afb37a246b797d902'
Submodule path 'dotfiles/dot/.vim/bundle/vim-airline-themes': checked out
'd148d42d9caf331ff08b6cae683d5b210003cde7'
Submodule path 'dotfiles/dot/.vim/bundle/vim-signify': checked out
'b2a0450e23c63b75bbeabf4f0c28f9b4b2480689'
```

Now if we later return to the repository and run `git submodule status --recursive` and get the following output

```
git submodule status --recursive

+7877117d5bd413ecf35c86efb4514742d8136843 dot (heads/master)
 826d5691ac7d36589591314621047b1b9d89ed34 dot/.vim/bundle/Colorizer (heads/master)
 3ea887d2f4d43dd55d81213517344226f6399ed6 dot/.vim/bundle/ale (v3.1.0-9-g3ea887d2)
 293a1062274a06be61797612034bd8d87851406e dot/.vim/bundle/clang_complete (v1.8-374-g293a106)
 d80e8e2c1fa08607fa34c0ca5f1b66d8a906c5ef dot/.vim/bundle/supertab (2.1-40-gd80e8e2)
 afb8db4f81580771c39967e89bc5772e72b9018e dot/.vim/bundle/unicode.vim (v20-139-gafb8db4)
 cb1bc19064d3762e4e08103afb37a246b797d902 dot/.vim/bundle/vim-airline (v0.11-354-gcb1bc19)
 d148d42d9caf331ff08b6cae683d5b210003cde7 dot/.vim/bundle/vim-airline-themes (remotes/origin/jellybeans-
refactor-266-gd148d42)
 b2a0450e23c63b75bbeabf4f0c28f9b4b2480689 dot/.vim/bundle/vim-signify (v1.0-291-gb2a0450)
```

This means the `dot` project submodule has a new commit that we haven't pulled into our local project yet. To fix this, just run the command below again.

```
git submodule update --init --recursive
```

Fixing or replacing submodules can be done with the following steps.

For context, see the status of all submodules

```
git submodule status

 826d5691ac7d36589591314621047b1b9d89ed34 .vim/bundle/Colorizer (heads/master)
 3ea887d2f4d43dd55d81213517344226f6399ed6 .vim/bundle/ale (v3.1.0-9-g3ea887d2)
 293a1062274a06be61797612034bd8d87851406e .vim/bundle/clang_complete (v1.8-374-g293a106)
 d80e8e2c1fa08607fa34c0ca5f1b66d8a906c5ef .vim/bundle/supertab (2.1-40-gd80e8e2)
 afb8db4f81580771c39967e89bc5772e72b9018e .vim/bundle/unicode.vim (v20-139-gafb8db4)
 cb1bc19064d3762e4e08103afb37a246b797d902 .vim/bundle/vim-airline (v0.11-354-gcb1bc19)
 d148d42d9caf331ff08b6cae683d5b210003cde7 .vim/bundle/vim-airline-themes (remotes/origin/jellybeans-
 refactor-266-gd148d42)
 b2a0450e23c63b75bbeabf4f0c28f9b4b2480689 .vim/bundle/vim-signify (v1.0-291-gb2a0450)
```

So we try to add a submodule, one that we had removed from .gitmodules and deleted the directory. But we get the error below.

```
git submodule add https://github.com/alexanderjeurissen/ranger_devicons
.config/ranger/plugins/ranger_devicons


A git directory for '.config/ranger/plugins/ranger_devicons' is found locally with remote(s):
  origin        https://github.com/alexanderjeurissen/ranger_devicons
If you want to reuse this local git directory instead of cloning again from
  https://github.com/alexanderjeurissen/ranger_devicons
use the '--force' option. If the local git directory is not the correct repo
or you are unsure what this means choose another name with the '--name' option.
```

To correct the awkward start our git modules are in, we simply remove all traces of the submodule. To do this, run the following commands

```
git rm -r --cached .config/ranger/plugins/ranger_devicons/
rm -r .config/ranger/plugins/ranger_devicons/
rm -rf .git/modules/.config/ranger/plugins/ranger_devicons/
vim .gitmodules
vim .git/config
```

The first file opened by the vim command is .gitmodules within the root of your repository. REMOVE the following lines, and save the file.

```
# .gitmodules file in the root of your repository
[submodule ".config/ranger/plugins/ranger_devicons"]
  path = .config/ranger/plugins/ranger_devicons
  url = https://github.com/alexanderjeurissen/ranger_devicons
```

The second file opened by the `vim` command is `.git/config`. REMOVE the following lines, and save the file.

```
# .git/config from the root of your repository
[submodule ".config/ranger/plugins/ranger_devicons"]
  url = https://github.com/alexanderjeurissen/ranger_devicons
  active = true
```

Now you can add the submodule back again

```
git submodule add https://github.com/alexanderjeurissen/ranger_devicons
.config/ranger/plugins/ranger_devicons

Cloning into '/home/kapper/dot/.config/ranger/plugins/ranger_devicons'...
remote: Enumerating objects: 329, done.
remote: Counting objects: 100% (91/91), done.
remote: Compressing objects: 100% (84/84), done.
remote: Total 329 (delta 37), reused 10 (delta 7), pack-reused 238
Receiving objects: 100% (329/329), 183.95 KiB | 2.11 MiB/s, done.
Resolving deltas: 100% (146/146), done.
```

```
git submodule status

 feb2d7a90fe8aabd7ee3965d4bd67ebedceca817 .config/ranger/plugins/ranger_devicons (heads/main)
 826d5691ac7d36589591314621047b1b9d89ed34 .vim/bundle/Colorizer (heads/master)
 3ea887d2f4d43dd55d81213517344226f6399ed6 .vim/bundle/ale (v3.1.0-9-g3ea887d2)
 293a1062274a06be61797612034bd8d87851406e .vim/bundle/clang_complete (v1.8-374-g293a106)
 d80e8e2c1fa08607fa34c0ca5f1b66d8a906c5ef .vim/bundle/supertab (2.1-40-gd80e8e2)
 afb8db4f81580771c39967e89bc5772e72b9018e .vim/bundle/unicode.vim (v20-139-gafb8db4)
 cb1bc19064d3762e4e08103afb37a246b797d902 .vim/bundle/vim-airline (v0.11-354-gcb1bc19)
 d148d42d9caf331ff08b6cae683d5b210003cde7 .vim/bundle/vim-airline-themes (remotes/origin/jellybeans-refactor-266-gd148d42)
 b2a0450e23c63b75bbeabf4f0c28f9b4b2480689 .vim/bundle/vim-signify (v1.0-291-gb2a0450)
```