Table of Contents

Introduction

Git

Git is a wonderful tool for source control management (scm), especially when working on multiple features as an individual or on a team.

Personally, I don't like to get too complicated with my git commands. I prefer to know the core of git well enough to serve me for 99% of of uses. Then using Google-fu for those 1% situations.

Terminology

Term	Description
local	on the harddrive of the computer running your git commands
remote	central scm server (i.e. Github, Gitlab, Bitbucket, etc.)
origin	traditional tag associated with a repo's remote
commit	set of file changes with a message (think of this similar to a checkpoint save)
branch	...
merge	...
rebase	...
push	upload your local commits to a remote
pull	download remote commits to your local
clone	download a remote repo to your local for the first time
staged	local files which have been prepared to be included in the next commit (the targets of `git add`)
unstaged	local files which have been included in a prior commit with changes not yet ready to be committed
untracked	local files within a repo's directory which have never been included in a commit

Commands

Set up ssh key for use with ssh git auth

```
ssh-keygen -t rsa -b 4096 -C "yourEmail@domain.com"
# provide optional passphrase
eval "$(ssh-agent -s)"
ssh-add -K ~/.ssh/id_rsa

# copy public key to scm provider
# follow their instructions for where you set in your auth profile
cat ~/.ssh/id_rsa.pub
```

Start a new local repo

```
mkdir localRepo
cd ./localRepo
git init
```

Pull Remote Repo to Local (for the first time)

```
# assuming Github project fakerepo, owned by dlstadther
# with ssh auth
git clone git@github.com:dlstadther/fakerepo.git

# with https basic auth
git clone https://github.com/dlstadther/fakerepo.git

# ssh auth with custom directory name
git clone git@github.com:dlstadther/fakerepo.git foobar
```

Update local references to remotes

```
git fetch origin
```

Start new branch

```
# create and checkout a new branch "feature/my-feature" based off local master
git checkout -b feature/my-feature master
```

View all locally tracked branches

```
git branch -a
```

Change branches

```
# assumes already on master, and the existence of develop
git checkout develop

# switch back to previous branch
git checkout -
```

Update branch with remote master

```
# without updating local master w/o merge commit
git fetch origin
git rebase origin/master

# with local update and merge commit
git checkout master
git pull origin master
git checkout -
git merge master
```

Update [feature] branch with remote

```
# with merge commit
git pull origin my-branch

# without merge commit
git pull --rebase
```

Rename branch

```
git branch -m oldname newname

# rename current branch
git branch -m newname
```

Trash all tracked (and uncommitted) changes

```
git reset --hard
```

Trash uncommitted changes for a single tracked file

```
git checkout HEAD -- path/to/file
```

Avoid whitespace issues

```
git diff --check
```

Blaming

Basic

```
git blame -L 24,48 mydir/myfile.py
```

Ignoring the impact of a commit (e.g. last commit was just formats)

```
git blame --ignore-rev {commit-hash} -L 24,48 mydir/myfile.py
```

Ignoring the impact of lots of commits (via file)

```
cat <EOF > .git-blame-ignore-revs
# Applied black
{commit-hash-full-40-char}

# PEP8 updates
{commit-hash-full-40-char}
EOF

git blame --ignore-revs-file .git-blame-ignore-revs -L 24,48 mydir/myfile.py
```

Update git config with name of file to always use for ignores

```
git config blame.ignoreRevsFile .git-blame-ignore-revs
git blame -L 24,48 mydir/myfile.py  # ignores specified changes
```

Important Links and Practices

General: https://github.com/MikeMcQuaid/GitInPractice/blob/master/12-CreatingACleanHistory.adoc

Use gitflow

Gitflow is a branching and development practice which encourages project organization and release management.

While I like the gitflow branching model, I do not like to rely on the obfuscation of the gitflow commands. Although requiring more keystrokes, I prefer to be hyper-aware of the component git commands which follow the gitflow branching model. Check out this gist for a comparison of the git equivalents of gitflow commands.

Write useful commit messages

Commit messages contain a lot of value if used well and written effectively. I strongly recommend reading Chris Beam's blog post about How to Write a Git Commit Message.

Consult git in practice

A very thorough and practical approach to teaching in-depth git: GitInPractice.

Bookmark git flight rules

Sometimes things go wrong and you need to know how to dig yourself out of a hole. In these moments, consult git-flight-rules.

Read documentation

When all else fails, or you have ample curiosity (and time), checkout the git manpages.