Version Control
with Git Cheat Sheet

add ➡   commit ➡   push ➡

| Working directory | Stage (index) | Local history | Remote history *origin* |

⟵ checkout   ⟵ pull
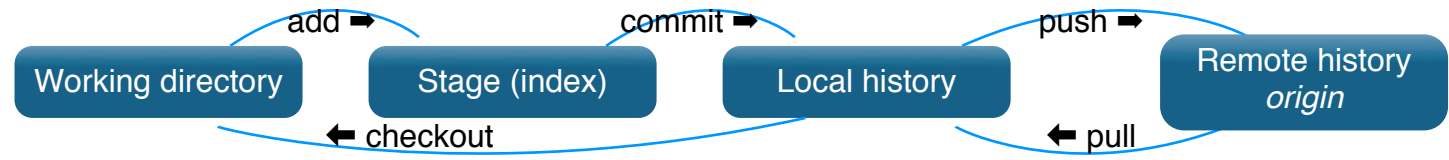
Mike Flow* Single workflow
1. **Create, checkout branch**
2. **Code, stage, commit a lot**
3. **Ready? Rebase into atomic commit**
4. **Merge branch into master**
5. **Tag it, push, delete branch**

Create

git clone <url>
From remote history
git clone git@github.com:mikec/myproj.git

git init; git add .;
git commit --message "initial commit"
Create local history from existing files in current directory

git remote add origin <url>;
git push --set-upstream origin master
From local history to blank remote history
git remote add origin git@github.com:mikec/myproject.git

git remote --verbose # verify origin url

Branch

git branch [--all | --verbose]
List branches. --all shows local and remotes

git branch <new-branch> [src-branch]
Create branch locally based on HEAD or src.

git checkout <branch1>
Switch to branch

git branch --move <branch1> <branch2>
Rename branch1 to branch2

git branch --delete <branch1>
Delete branch 1

git stash save | pop
Save leaves branch without commit, Pop restores branch's unsaved work

Add/Reset

git add .|--update | <files>
Add all new/modified/deleted or specified files to stage. --update skips new files.

git mv <files>
Rename or move files; update stage

git rm [--cached] <files>
Delete file from working area and index. --cached removes from history only.

git reset [files]
Remove all (or specified) files from stage without changing working directory.

git clean [--dry-run | --force]
DANGER! Deletes unstaged files.

git reset --hard
DANGER! Deletes uncommited files.

Work/Review

git status
Files changed in working directory

git diff [file]   # working v stage code changes
Show code changes between working and stage

git diff HEAD   # working v last commit code changes
Show code changes between working and last commit

git diff --cached # stage v last commit code changes
Show code changes between stage and last commit

git diff <commit1> <commit2> # history code changes

git log [--oneline | --graph | --decorate]
History of commits

git log <branch> **--not master**
History of commits for branch

git reflog [--relative-date | --all]
Show changes to local HEAD.

git show <commit>[:<file>]
Show commit and code changes. :file narrows scope

Commit/Revert

git commit [--all | --message "<description title>"]
Commit with only description title instead of launching editor for full description

Commit description: title, blank line, body

git commit --amend
Combine new changes with last commit, overwrite last description

git revert HEAD | <commit>
Revert last commit; create new commit

git checkout <commit> <file>
Revert last commit; create new commit

git rebase --interactive master
Clean up/combine commits and modify history of current branch against master. Edit 1st word of each line, usually with fixup.

Resolve Merge Conflicts

git merge <branch1> [branch2]
Merge branch1 into current branch

git merge --no-ff [branch1]
Merge without fast-forward to create a merge commit. This aids history visualization.

Git will try resolve a 3-way merges. Successes will be staged, conflicts will be unmerged. Use git status to list them.

git diff [--base | --ours | --theirs] <file>
Compares file to base (base) file, your (ours) changes, other (theirs) changes, or all.

git mergetool <file>
Launches previously configured GUI mergetool. Make appropriate changes, then confirm at CLI.

Push/Pull

git tag [--annotate | -n] <tag>
Tags current branch. -n lists tags annotations

git push --set-upstream origin <branch>
Creates upstream branch, sends branch to remote history (origin)

git push origin [branch] [--all | --tags]
Pushes current branch, if upstream branch exists. --all pushes all branches, --tags pushes tags.

git push origin --all --tags;
   git push --delete origin <branch>;
   git branch --delete <branch>
Push all branches, delete branch on remote origin, delete local branch

git pull origin <branch>
Get changes from remote and merge

git fetch origin <branch>
Get changes from remote without merge

git cherry-pick <commit>
Brings changes (not all files) from commit to working directory

1. Identify which files have merge conflicts with git status.
2. Manually resolve conflicts in each file with **vim** or **mergetool**, look for <<<< through >>>>.

```
unchanged code for context
<<<< HEAD (current branch marker)
current code
==== (branch separator)
incoming code
>>>> branch-name (incoming branch marker)
```

3. Chose current or incoming code or merge the contents, then delete markers and separator.
4. Use git add for resolved file, delete .orig file.
5. Use git commit when all files are resolved.

Git Basics

The Feature Branch Workflow assumes a new branch for every new feature. The master branch never has broken code.

Feature branches are pushed to the repository. Commits should be atomic: Self-contained and no code should span multiple commits.

The Mike Flow workflow is from Mike McQuaid's book **Git In Practice**. It's modified GitHub Flow but branches are Stable releases are tagged on the master branch.

The local repository is called *history* and the remote repository is called *remote history* (or *origin*, its most common name). The default branch name in a new history is **master**. A remote tracking branch (like *origin/master*) is called an upstream branch.

HEAD: current commit (and branch)
<commit>: HEAD, tag name, branch name, or leading substring of the commit SHA-1
<commit>^: 1st parent of commit
<commit>~2: nth gen ancestor (grandparent) of commit (following only 1st parents)
<file>: filespec
<branch>: branch names cannot contain spaces

exact <required> [optional] choice | alternate
repeatable…
  Explanation of the code
  # example code

Command options are shown in long form for better mnemonics; --message instead of -m

Don't forget: **git help** [command]

Configure

git config --global user.name "Mike Combs"

git config --global user.email "mike@combsnet.com"

git config --global core.editor "vim"

git config --global merge.tool "diffmerge"

git config --global credential.helper osxkeychain
  Use keychain for passwords instead of reprompting. Also: git-credential-gnome-keyring or git-credential-winstore

git config --global --edit
  Open ~/.gitconfig global config file in editor for editing

vim .gitignore
  Edit this to ignore temporary, object, project, and other files. Find examples at https://github.com/github/gitignore. For example:

```
.Rhistory
.RData
.Rproj.user/
.DS_Store
__pycache__/
*.py[cod]
*$py.class
```

git rm --cached <files>
  Remove files from history in case they got there before you put them in .gitignore

git config --global alias.<name> <cmd>
  Define an alias for <cmd>
  git config --global alias.log1 "log --graph --decorate --oneline"

git <alias>
  Use alias previously defined

Log Options

git log <since id>..<until id> # show range of commits

git log -<limit> # limit number of commits shown

git log --author="<pattern>"

git log --decorate # show branch and tag names

git log --graph # show graph of commits

git log --grep="<pattern>"

git log --oneline # show each commit on 1 line

git log -p # show full diff

git log --stat # show files and changed line counts

Regression

git blame --date=short -w -s -L 40,60 <file>
  For each line in file, show author, date, and commit. -w ignores white space, -s hides author name, -L specifies range of lines

Bisect helps find a commit that introduced a bug through a binary search of history.

git bisect start; git bisect bad;
  git bisect good <commit>
  Start regression process, indicate this commit is bad, identify last known good commit. Bisect will now checkout a revision within these bounds.

git bisect <bad | good>
  User should check for problem, then use this to mark revision and checkout the next one. Repeat until
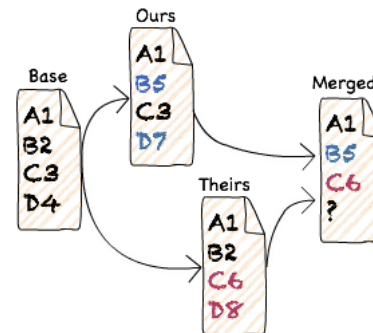
git bisect reset
  End regression process, return to HEAD
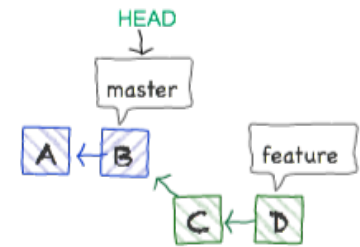
git bisect log
  Show bisect steps

3-Way Merge

In this merge, Git cannot automatically resolve line D because it has been changed in two branches. Usually *ours* is the current and master branch, and *theirs* is the feature branch.
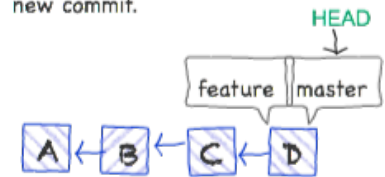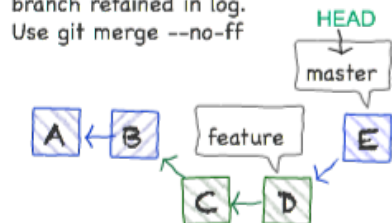

FF-Merge, Merge Commit

BEFORE merge: master & feature branches, no changes to master

AFTER fast-forward merge: Default if no changes to master. No new commit.

AFTER merge commit: New commit, branch retained in log. Use git merge --no-ff

References

Git in Practice — Mike McQuaid, Manning Publications

https://git-scm.com/docs

https://www.atlassian.com/git/tutorials

http://ndpsoftware.com/git-cheatsheet.html

https://backlog.com/git-tutorial/