# Common Git Commands

## Git Crash Course

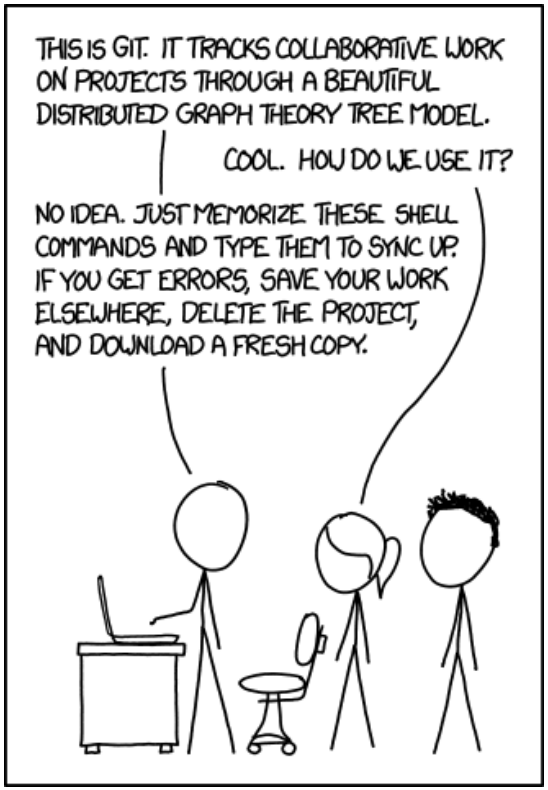
Teon Banek
theongugl@gmail.com


April 7, 2016

# Outline

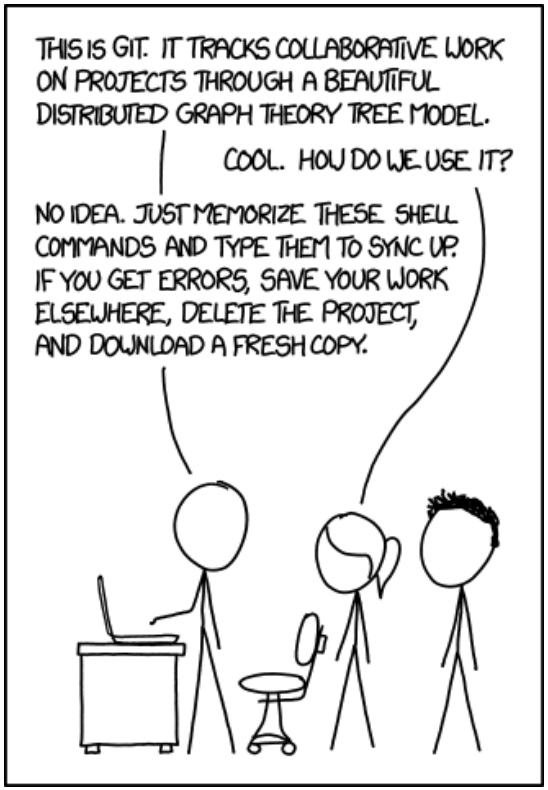
# What is Git?

## The stupid content tracker.



# What is Git?

## The stupid content tracker.




*If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of 'It's really pretty simple, just think of branches as...' and eventually you'll learn the commands that will fix everything.*

# Using Git

- Command line

# Using Git

- Command line
  - Linux – Use your favorite shell.
  - Windows – `cmd` may and will break; use Git Bash.

# Using Git

- Command line
  - Linux – Use your favorite shell.
  - Windows – `cmd` may and will break; use Git Bash.
- Browsing history

# Using Git

- Command line
  - Linux – Use your favorite shell.
  - Windows – `cmd` may and will break; use Git Bash.
- Browsing history
  - git log – Powerful, but not user friendly
  - tig – Console based, yet graphical; recommended.
  - gitk – GUI, recommended on Windows.

# Using Git

- Command line
  - Linux – Use your favorite shell.
  - Windows – `cmd` may and will break; use Git Bash.
- Browsing history
  - git log – Powerful, but not user friendly
  - tig – Console based, yet graphical; recommended.
  - gitk – GUI, recommended on Windows.
- Editor plugins

# Using Git

- Command line
  - Linux – Use your favorite shell.
  - Windows – `cmd` may and will break; use Git Bash.
- Browsing history
  - git log – Powerful, but not user friendly
  - tig – Console based, yet graphical; recommended.
  - gitk – GUI, recommended on Windows.
- Editor plugins
  - Emacs – Magit; best Git integration in any editor.
  - Vim – Fugitive; it's great.
  - IDEs – Most have integration, usually terrible, prefer command line.

# Playground Setup

- Git User Setup

```
git config --global user.name "John Doe"
git config --global user.email "user@domain"
git config --global core.editor "/path/to/my/editor"
```

- Directory Setup

```
mkdir git-playground
cd git-playground
git init local
git init --bare server.git
cd local
```

# States

1. Working tree – your local files.
2. Index (staging area) – files ready for commit.
3. Repository – commits.

# States

1. Working tree – your local files.
   - Create/Edit: `code.py`, `private.txt`, `.gitignore`
   - `git status`
   - `git diff`
2. Index (staging area) – files ready for commit.
   - `git add`
3. Repository – commits.
   - `git commit`

# States

1. Working tree – your local files.
   - Create/Edit: `code.py`, `private.txt`, `.gitignore`
   - `git status`
   - `git diff`
2. Index (staging area) – files ready for commit.
   - `git add`
   - `git reset`
3. Repository – commits.
   - `git commit`
   - `git reset --soft HEAD^`

# Remotes

- List and add remote repositories.
  - `git remote -v`
  - `git remote add <name> <url>`

# Remotes

- List and add remote repositories.
  - `git remote -v`
  - `git remote add <name> <url>`
- Push to remote.
  - `git push <remote> <local-branch>`
- Pull from remote.
  - `git pull <remote> <remote-branch>`

# Remotes

- List and add remote repositories.
  - `git remote -v`
  - `git remote add <name> <url>`
- Push to remote.
  - `git push <remote> <local-branch>`
- Pull from remote.
  - `git pull <remote> <remote-branch>`
- Clone a repository.
  - `git clone <url> <directory>`

# Branching

- List, create and delete branches.
  - `git branch`
  - `git branch <name>`
  - `git branch -d <name>`

# Branching

- List, create and delete branches.
  - `git branch`
  - `git branch <name>`
  - `git branch -d <name>`
- Switch branches.
  - `git checkout <name>`

# Branching

- List, create and delete branches.
  - `git branch`
  - `git branch <name>`
  - `git branch -d <name>`
- Switch branches.
  - `git checkout <name>`
- Create and switch to branch.
  - `git checkout -b <name>`

# Branching

- List, create and delete branches.
  - `git branch`
  - `git branch <name>`
  - `git branch -d <name>`
- Switch branches.
  - `git checkout <name>`
- Create and switch to branch.
  - `git checkout -b <name>`
- Differences between branches.
  - `git diff <src>..<dst>`

# Merging

- Merge named branch into current.
  - `git merge <name>`

# Merging

- Merge named branch into current.
  - `git merge <name>`
- Resolve conflicts or abort.
  - `HEAD` vs `branch-name`
  - `git merge --abort`
  - `git add <resolved-file>`
  - `git commit`

# Merging

- Merge named branch into current.
  - `git merge <name>`
- Resolve conflicts or abort.
  - `HEAD` vs `branch-name`
  - `git merge --abort`
  - `git add <resolved-file>`
  - `git commit`
- Avoiding non fast forward merges.
  - `git rebase <onto> <from>`
  - `git rebase --abort`
  - `git rebase --continue`

# Merging

- Merge named branch into current.
  - `git merge <name>`
- Resolve conflicts or abort.
  - `HEAD` vs `branch-name`
  - `git merge --abort`
  - `git add <resolved-file>`
  - `git commit`
- Avoiding non fast forward merges.
  - `git rebase <onto> <from>`
  - `git rebase --abort`
  - `git rebase --continue`
- Forcing non fast forward merges.
  - `git merge --no-ff`

# Interactive Rebase

- `git rebase -i <onto> <from>`
- Very powerful tool; be very careful!
- Don't use it on branches where multiple people work.

# Interactive Rebase

- `git rebase -i <onto> <from>`
- Very powerful tool; be very careful!
- Don't use it on branches where multiple people work.
- Common uses
  - Reordering commits

# Interactive Rebase

- `git rebase -i <onto> <from>`
- Very powerful tool; be very careful!
- Don't use it on branches where multiple people work.
- Common uses
  - Reordering commits
  - Merging commits

# Interactive Rebase

- `git rebase -i <onto> <from>`
- Very powerful tool; be very careful!
- Don't use it on branches where multiple people work.
- Common uses
  - Reordering commits
  - Merging commits
  - Rewording commits

# Interactive Rebase

- `git rebase -i <onto> <from>`
- Very powerful tool; be very careful!
- Don't use it on branches where multiple people work.
- Common uses
  - Reordering commits
  - Merging commits
  - Rewording commits
  - Splitting commits

# Interactive Rebase

- `git rebase -i <onto> <from>`
- Very powerful tool; be very careful!
- Don't use it on branches where multiple people work.
- Common uses
  - Reordering commits
  - Merging commits
  - Rewording commits
  - Splitting commits
  - Deleting commits

# Great Power, Great Responsibility

- I made a mistake and now my work is gone, what do I do?

# Great Power, Great Responsibility

- I made a mistake and now my work is gone, what do I do?
- Don't panic!

# Great Power, Great Responsibility

- I made a mistake and now my work is gone, what do I do?
- Don't panic!
- Git is garbage collected
  - Automatically called after git commands
  - Called in intervals or when low on space

# Great Power, Great Responsibility

- I made a mistake and now my work is gone, what do I do?
- Don't panic!
- Git is garbage collected
  - Automatically called after git commands
  - Called in intervals or when low on space
- `git reflog` to the rescue
  - `--grep=<pattern>` – search by log messages.
  - `-S<string>` – search by source code changes.

# Great Power, Great Responsibility

- I made a mistake and now my work is gone, what do I do?
- Don't panic!
- Git is garbage collected
  - Automatically called after git commands
  - Called in intervals or when low on space
- `git reflog` to the rescue
  - `--grep=<pattern>` – search by log messages.
  - `-S<string>` – search by source code changes.
- Show and apply a commit
  - `git show` – show the reference object.
  - `git cherry-pick` – apply the referenced commit.

# Pushing a Reordered Branch

- After reordering the history, performing a plain push will not work.

# Pushing a Reordered Branch

- After reordering the history, performing a plain push will not work.
- `git push --force`
  - Overwrites the remote branch with your local.
  - Be extremely careful and warn your colleagues!

# Pushing a Reordered Branch

- After reordering the history, performing a plain push will not work.
- `git push --force`
  - Overwrites the remote branch with your local.
  - Be extremely careful and warn your colleagues!
- Regular pull will also not work.

# Pushing a Reordered Branch

- After reordering the history, performing a plain push will not work.
- `git push --force`
  - Overwrites the remote branch with your local.
  - Be extremely careful and warn your colleagues!
- Regular pull will also not work.
- `git fetch`
- `git rebase`
  - Applies your changes after the remote branch.

# Useful Commands

- `git cherry-pick`
  - Pick a commit from anywhere and apply it.
  - Very useful for applying fixes to stable branches.

# Useful Commands

- `git cherry-pick`
  - Pick a commit from anywhere and apply it.
  - Very useful for applying fixes to stable branches.
- `git stash`
  - Store work in progress changes.
  - Useful for when you need to pause what ever you are doing.
  - `pop`, `show`, `list`

# Finding Bugs

- `git bisect`
  - Perform a binary search on commits.
  - Very useful for tracking a change which introduced a bug.
  - `start`, `reset`
  - `good`, `bad`, `skip`,

# Finding Bugs

- `git bisect`
  - Perform a binary search on commits.
  - Very useful for tracking a change which introduced a bug.
  - `start`, `reset`
  - `good`, `bad`, `skip`,
- `git blame <file>`
  - Find the culprit.
  - Easier to use from other tools, like tig and gitk.

# Purging a File

- Git stores every change in each repository.
  - Both a virtue and a con of distributed version control.
  - Sometimes you want to completely delete a file, e.g., when the file takes unnecessary space or you committed private data.

# Purging a File

- Git stores every change in each repository.
  - Both a virtue and a con of distributed version control.
  - Sometimes you want to completely delete a file, e.g., when the file takes unnecessary space or you committed private data.
- `git filter-branch --force --index-filter 'git rm --cached --ignore-unmatch <file>' --prune-empty --tag-name-filter cat -- --all`
  - `--index-filter` – rewrite the index.
  - `git rm` – remove the `<file>` from commits *and your local disk*.
  - `--prune-empty` – don't leave empty commits.
  - `--tag-name-filter cat` – update tags.
  - `-- --all` – filter all branches.
  - All of history is rewritten, which means you will need to perform a force push.

# Purging a File

- After you are sure you didn't delete any other files, you need to clear the ref log.

# Purging a File

- After you are sure you didn't delete any other files, you need to clear the ref log.
  - `git reflog expire --expire=now --all`
  - `git gc --prune=now`

# The End

# The End

- Anything you think of can be done with git.

# The End

- Anything you think of can be done with git.
- If you are not sure about something, contact your nearest git expert.

# The End

- Anything you think of can be done with git.
- If you are not sure about something, contact your nearest git expert.
- Git is not the silver bullet, if you don't like it you can try:
  - Mercurial
  - Darcs
  - Fossil
  - Subversion

# Questions?

- Thank you for your attention!