Introduction to Version Control + Git

01 Oct 2014

Tim Skirvin <tskirvin@fnal.gov>

USCMS-T1 @ FNAL

What is Version Control?

- Structured way of tracking changes to a set of files
- Core functionality:
  - Primarily works with text or source code
  - Mark files as ‘managed’
  - Commit changes when you’re ready
    - …with some level of conflict detection/resolution
  - View a log of what has changed
  - Revert to older versions of files when necessary

Why do I need version control?

- Almost required to work with others
  - …or your own future self!
  - Multiple people can change the same file fairly safely
- Lets you revert past mistakes
- Commit messages provide useful meta-data regarding what you’re doing, when, and why
- You can work off-line or under low-bandwidth circumstances
- You can track changes over time.

Version Control through the Ages

- RCS (1982)
- CVS (1986) (still used occasionally)
- Subversion (2000) (still used extensively)
- git (2005)

- Many others: bitkeeper, mercurial (hg), bzr, etc
- Changing from one to another is a project!

Distributed Version Control

- pre-distributed VCS are client-server - have a single "master”
- distributed VCS allow many copies of the source repository
  - Each user has a full copy of the history
  - You can still have a "master", but it's only by convention
- Advantages:
  - You can work off-line
  - Most operations don’t require talking to the server -> faster
  - Makes it easy to fork off new versions of existing software
  - Easier to build a variety of support tools
    - Server: gitolite, github
    - Client: gitk
  - Easy to work with multiple masters as well

Git

- Distributed Version Control System (VCS)
- Originally created to manage the Linux kernel
  - Lots of potential branches
  - Very fast
  - Works everywhere
- github.com – social network for code
  - Other options: redmine (local), bitbucket, etc
- Currently winning the "VCS War"

How Git Works

```
daishi ~/puppet% ls
Makefile    enc@              hieradata/  meetings@  profiles/  tools/
Puppetfile  environment.conf  manifests/  modules/   test.pp    uscms_t1/

daishi ~/puppet% ls .git
COMMIT_EDITMSG  HEAD       config       hooks/  info/  objects/     refs/
FETCH_HEAD      ORIG_HEAD  description  index   logs/  packed-refs
```

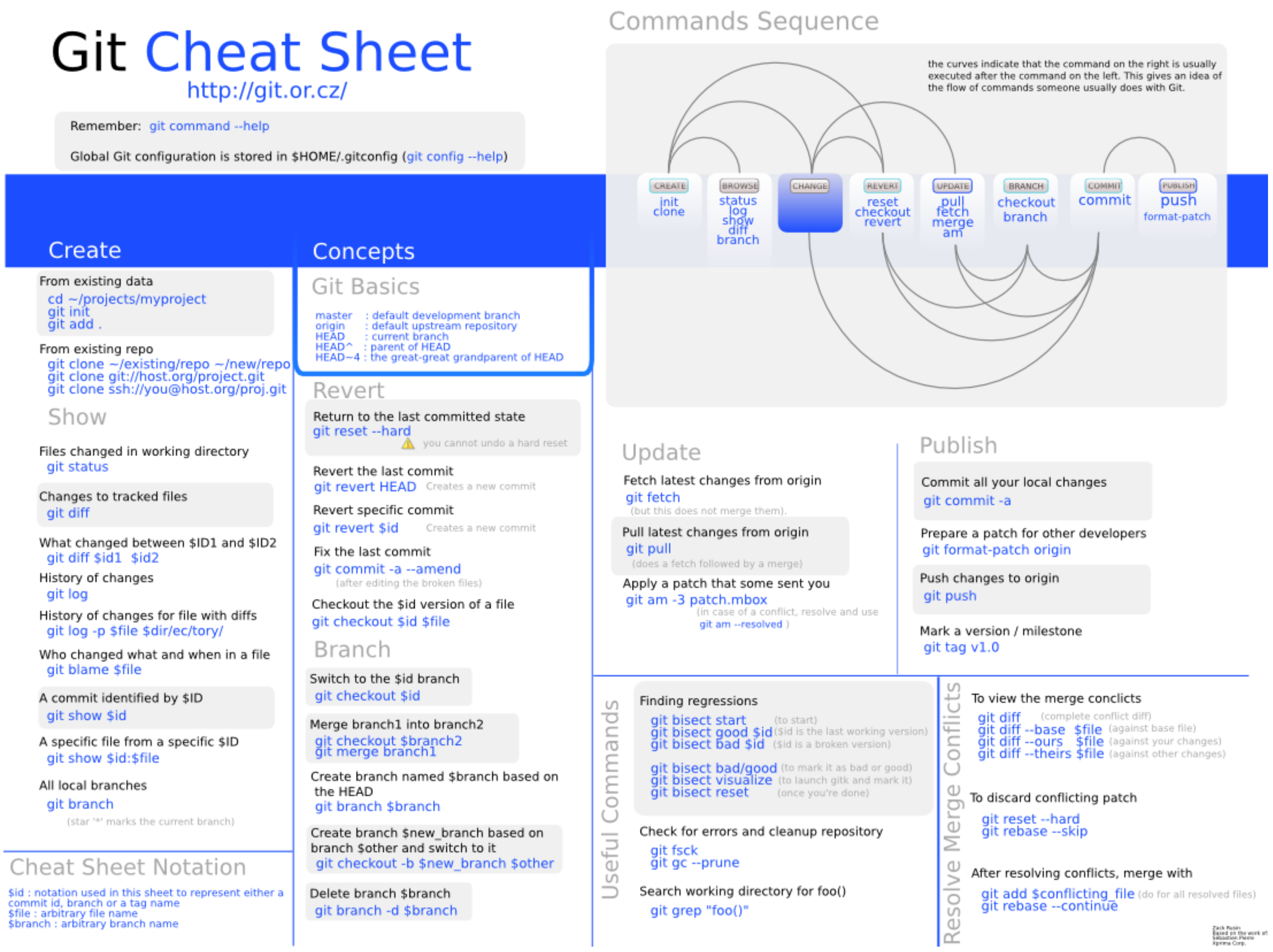
- Top-level directory – normal list of files
- `.git` directory – the git side of things
  - .git/config – configuration options, including lists of branches, remote repositories, aliases, etc
  - .git/hooks/* - scripts to run after making changes
  - .git/index, .index/objects/* - actual data
  - .git/HEAD – tag pointing at the current version
- You will rarely have to work in .git directly!

Git Commands

- ‘man git-clone’, ‘man git-init’
  - Real help comes from Google
- Initialization: git clone, git init
- Local Changes: git add, git commit, git rm, git mv
- Check Status: git status, git log, git diff
- Remote Changes: git push, git pull
- Branches: git branch, git checkout –b
- Revert Changes: git revert

Core Workflow

1. (Initialize repo)
2. Make Changes
3. Commit Changes Locally
4. Push Changes to Remote Server



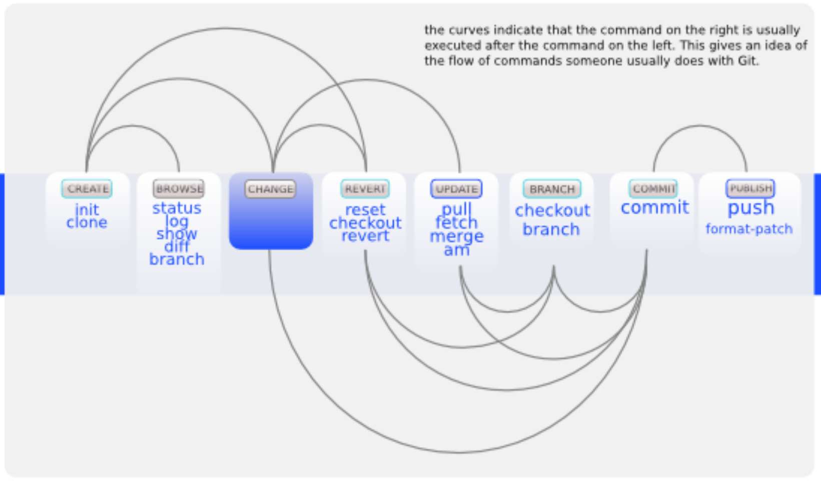
Git Cheat Sheet

http://git.or.cz/

Remember: `git command --help`

Global Git configuration is stored in $HOME/.gitconfig (`git config --help`)

Commands Sequence



the curves indicate that the command on the right is usually executed after the command on the left. This gives an idea of the flow of commands someone usually does with Git.
CREATE
init
clone
BROWSE
status
log
show
diff
branch
CHANGE
REVERT
reset
checkout
revert
UPDATE
pull
fetch
merge
am
BRANCH
checkout
branch
COMMIT
commit
PUBLISH
push
format-patch

Create

From existing data

```
cd ~/projects/myproject
git init
git add .
```

From existing repo

```
git clone ~/existing/repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://you@host.org/proj.git
```

Show

Files changed in working directory

```
git status
```

Changes to tracked files

```
git diff
```

What changed between $ID1 and $ID2

```
git diff $id1  $id2
```

History of changes

```
git log
```

History of changes for file with diffs

```
git log -p $file $dir/ec/tory/
```

Who changed what and when in a file

```
git blame $file
```

A commit identified by $ID

```
git show $id
```

A specific file from a specific $ID

```
git show $id:$file
```

All local branches

```
git branch
```

(star '*' marks the current branch)

Cheat Sheet Notation

$id : notation used in this sheet to represent either a commit id, branch or a tag name
$file : arbitrary file name
$branch : arbitrary branch name

Concepts

Git Basics

master	: default development branch
origin	: default upstream repository
HEAD	: current branch
HEAD^	: parent of HEAD
HEAD~4	: the great-great grandparent of HEAD

Revert

Return to the last committed state

```
git reset --hard
```

⚠ you cannot undo a hard reset

Revert the last commit

```
git revert HEAD
```

Creates a new commit

Revert specific commit

```
git revert $id
```

Creates a new commit

Fix the last commit

```
git commit -a --amend
```

(after editing the broken files)

Checkout the $id version of a file

```
git checkout $id $file
```

Branch

Switch to the $id branch

```
git checkout $id
```

Merge branch1 into branch2

```
git checkout $branch2
git merge branch1
```

Create branch named $branch based on the HEAD

```
git branch $branch
```

Create branch $new_branch based on branch $other and switch to it

```
git checkout -b $new_branch $other
```

Delete branch $branch

```
git branch -d $branch
```

Update

Fetch latest changes from origin

```
git fetch
```

(but this does not merge them).

Pull latest changes from origin

```
git pull
```

(does a fetch followed by a merge)

Apply a patch that some sent you

```
git am -3 patch.mbox
```

(in case of a conflict, resolve and use `git am --resolved`)

Publish

Commit all your local changes

```
git commit -a
```

Prepare a patch for other developers

```
git format-patch origin
```

Push changes to origin

```
git push
```

Mark a version / milestone

```
git tag v1.0
```

Useful Commands

Finding regressions

```
git bisect start         (to start)
git bisect good $id      ($id is the last working version)
git bisect bad $id       ($id is a broken version)

git bisect bad/good      (to mark it as bad or good)
git bisect visualize     (to launch gitk and mark it)
git bisect reset         (once you're done)
```

Check for errors and cleanup repository

```
git fsck
git gc --prune
```

Search working directory for foo()

```
git grep "foo()"
```

Resolve Merge Conflicts

To view the merge conclicts

```
git diff                (complete conflict diff)
git diff --base $file   (against base file)
git diff --ours   $file (against your changes)
git diff --theirs $file (against other changes)
```

To discard conflicting patch

```
git reset --hard
git rebase --skip
```

After resolving conflicts, merge with

```
git add $conflicting_file  (do for all resolved files)
git rebase --continue
```

Zack Rusin
Based on the work of:
Sébastien Pierre
Xprima Corp.

http://byte.kde.org/~zrusin/git/git-cheat-sheet-medium.png

git clone

```
daishi ~/tmp% git clone https://github.com/tskirvin/fnal-git-demo.git
Cloning into 'fnal-git-demo'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (3/3), done.
Checking connectivity... done
daishi ~/tmp% ls
fnal-git-demo/
daishi ~/tmp% ls fnal-git-demo
README.md
```

git init

```
daishi ~/tmp% mkdir test
daishi ~/tmp% cd test
daishi ~/tmp/test% touch testing
daishi ~/tmp/test% git init
Initialized empty Git repository in /Users/tskirvin/tmp/test/.git/
daishi ~/tmp/test% git add testing
daishi ~/tmp/test% git commit -m "initial commit"
[master (root-commit) b82507e] initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 testing
```

git add

```
daishi ~/tmp/test% echo "testing" > 1
daishi ~/tmp/test% touch 2
daishi ~/tmp/test% git add 1 2
daishi ~/tmp/test% git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   1
#       new file:   2
#
```

git commit

```
daishi ~/tmp/test% git commit -m "look at my changes, my changes are
amazing"
[master 0a8ca4a] look at my changes, my changes are amazing
 2 files changed, 1 insertion(+)
 create mode 100644 1
 create mode 100644 2
daishi ~/tmp/test% git status
# On branch master
nothing to commit, working directory clean
```

git mv, git rm

```
daishi ~/tmp/test% git mv 2 foo
daishi ~/tmp/test% git mv 1 bar
daishi ~/tmp/test% git rm testing
rm 'testing'
daishi ~/tmp/test% git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       renamed:    1 -> bar
#       renamed:    2 -> foo
#       deleted:    testing
#
daishi ~/tmp/test% git commit
     …writes something in the editor…
[master 15c6f0d] moving files around
 3 files changed, 0 insertions(+), 0 deletions(-)
 rename 1 => bar (100%)
 rename 2 => foo (100%)
 delete mode 100644 testing
```

git status

```
daishi ~/tmp/test% git status
# On branch master
```

git log

```
daishi ~/tmp/test% git log | cat
commit 15c6f0d86e1882962b8a9131bda1bdda0241986d
Author: Tim Skirvin <tskirvin@fnal.gov>
Date:   Mon Sep 29 13:17:27 2014 -0500

    moving files around

commit 0a8ca4ae8a66fa7537d746ecefd124752580dcf1
Author: Tim Skirvin <tskirvin@fnal.gov>
Date:   Mon Sep 29 13:15:07 2014 -0500

    look at my changes, my changes are amazing

commit b82507e3ff368bfb32a2183629fe71fb7a7a06f4
Author: Tim Skirvin <tskirvin@fnal.gov>
Date:   Mon Sep 29 13:08:38 2014 -0500

    initial commit
```

git diff

```
daishi ~/tmp/test% git diff 0a8ca4ae8a66fa7537d746ecefd124752580dcf1
| cat
diff --git a/1 b/1
deleted file mode 100644
index 038d718..0000000
--- a/1
+++ /dev/null
@@ -1 +0,0 @@
-testing
diff --git a/2 b/2
deleted file mode 100644
index e69de29..0000000
diff --git a/bar b/bar
new file mode 100644
index 0000000..038d718
--- /dev/null
+++ b/bar
@@ -0,0 +1 @@
+testing
diff --git a/foo b/foo
new file mode 100644
index 0000000..e69de29
diff --git a/testing b/testing
deleted file mode 100644
index e69de29..0000000
```

git push

```
daishi ~/tmp% git clone https://github.com/tskirvin/fnal-git-demo.git
2
Cloning into '2'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (3/3), done.
Checking connectivity... done
daishi ~/tmp% cd 2
daishi ~/tmp/2% touch baz
daishi ~/tmp/2% git add baz
daishi ~/tmp/2% git commit -m "bazzzzz"
[master 1eb67fd] bazzzzz
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 baz
daishi ~/tmp/2% git push
Username for 'https://github.com': tskirvin
Password for 'https://tskirvin@github.com':
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 267 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/tskirvin/fnal-git-demo.git
   d6ac38d..1eb67fd  master -> master
```

git pull

```
daishi ~/tmp/2% cd ../fnal-git-demo
daishi ~/tmp/fnal-git-demo% git status
# On branch master
nothing to commit, working directory clean
daishi ~/tmp/fnal-git-demo% git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://github.com/tskirvin/fnal-git-demo
   d6ac38d..1eb67fd  master     -> origin/master
Updating d6ac38d..1eb67fd
Fast-forward
 baz | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 baz
daishi ~/tmp/fnal-git-demo% ls
README.md  baz
daishi ~/tmp/fnal-git-demo% git log | head -5
commit 1eb67fdbc3572733c6708523758eaa6da8b2d1e0
Author: Tim Skirvin <tskirvin@fnal.gov>
Date:   Mon Sep 29 13:26:44 2014 -0500

    bazzzzz
```

git checkout, git branch

```
daishi ~/tmp/fnal-git-demo% git checkout -b testing
Switched to a new branch 'testing'
daishi ~/tmp/fnal-git-demo% touch 1
daishi ~/tmp/fnal-git-demo% git add 1
daishi ~/tmp/fnal-git-demo% git commit -m "temp file"
[testing 6267db9] temp file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1
daishi ~/tmp/fnal-git-demo% git checkout master
Switched to branch 'master'
daishi ~/tmp/fnal-git-demo% ls
README.md  baz

daishi ~/tmp/fnal-git-demo% git checkout testing
Switched to branch 'testing'
daishi ~/tmp/fnal-git-demo% ls
1  README.md  baz
```

git revert

```
daishi ~/tmp/fnal-git-demo% echo "fooo" > 3
daishi ~/tmp/fnal-git-demo% git add 3; git commit -m "333"
[master af28483] 333
 1 file changed, 1 insertion(+)
 create mode 100644 3
daishi ~/tmp/fnal-git-demo% ls
3  README.md  baz
daishi ~/tmp/fnal-git-demo% git log | head -5
commit af28483f1a1cd3c648b2e9cfc891344403af66ef
Author: Tim Skirvin <tskirvin@fnal.gov>
Date:   Mon Sep 29 13:34:02 2014 -0500

    333
daishi ~/tmp/fnal-git-demo% git revert
af28483f1a1cd3c648b2e9cfc891344403af66ef
    …into the editor again…
[master 317898c] Revert "333"
 1 file changed, 1 deletion(-)
 delete mode 100644 3
daishi ~/tmp/fnal-git-demo% git log | head -5
commit 317898c76e5badc5e97b2dae84a3ebcf3bf5c5b9
Author: Tim Skirvin <tskirvin@fnal.gov>
Date:   Mon Sep 29 13:34:36 2014 -0500

    Revert "333”
daishi ~/tmp/fnal-git-demo% ls
README.md  baz
```

git blame

```
daishi ~/tmp/fnal-git-demo% git blame README.md | cat
^d6ac38d (Tim Skirvin 2014-09-29 13:07:10 -0500 1) Hi there, Fermi folks!
```

Workflow: In-Place Repositories

- Useful when you just want to track revisions of an existing directory
- Example (as root):

```
cd /etc
git init
git add fstab
git commit –m “initial fstab”
# edit fstab
git add fstab
git commit –m “fstab – MY UPDATES”
```

There is **No Shame** in Asking for Help

- git is only simple at its core
  - Manual pages are not always easy to follow
- There are many potential workflows
- Merging branches is especially tricky

- Google for help
- Ask your local experts
- Ask on linux-users
- Maybe make a local git-users list?

Advice

- Always write descriptive commit messages
  - The first line of text should be a good summary
  - More complicated commits can include additional paragraphs of text
- http://git-scm.com/book/en/Git-Basics-Undoing-Things
- Don’t use ‘`git commit –a`’
- Try things out in an new branch
- If everything has gone badly, don’t be afraid to start over with a new clone of the repo

Next Steps

- Amit will now do a live git walkthrough
- Marc will talk about local git resources and how to use them

- We'll all take questions
- Let's all think about future training/ documentation needs here at FNAL
- My favorite git video: http://www.oscon.com/oscon2011/public/schedule/detail/18768 ("git for ages 4 and up")